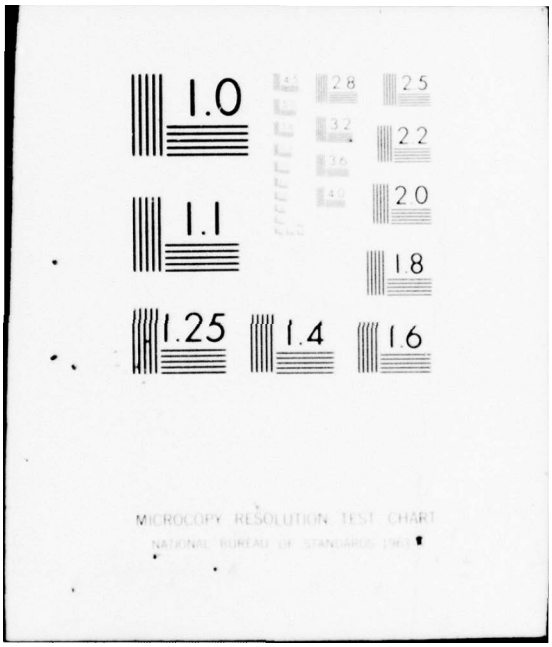
AD-A034 228 BDM CORP EL PASO TEX F/G 9/2
DATA BASE MANAGEMENT SYSTEM REFERENCE MANUAL.(U)
OCT 76 J M PHELAN
UNCLASSIFIED AFWL-TR-75-159 F29601-74-C-0017 NL

1 OF 2
AD A034228


1.0
1.1
1.25
1.4
1.6
2.8
2.5
3.2
2.2
3.6
2.0
4.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963

AFWL-TR-75-159

AFWL-TR-
75-159

DATA BASE MANAGEMENT SYSTEM REFERENCE MANUAL

BDM Corporation
6070 Gateway East
El Paso, TX 79925

October 1976

Final Report

This research was sponsored by the Defense Nuclear Agency under Subtask Z99QAXTC022, Work Unit 52, Work Unit Title: Interface Program for Circuit Analysis.

Prepared for
Director
DEFENSE NUCLEAR AGENCY
Washington, DC 20305

AIR FORCE WEAPONS LABORATORY
Air Force Systems Command
Kirtland Air Force Base, NM 87117

AFWL-TR-75-159

This final report was prepared by the BDM Corporation, El Paso, TX, under Contract F29601-74-C-0017, Job Order WDNE1308, with the Air Force Weapons Laboratory, Kirtland AFB, NM. Mr. A. Brent White (ELP) was the Laboratory Project Officer-in-Charge.

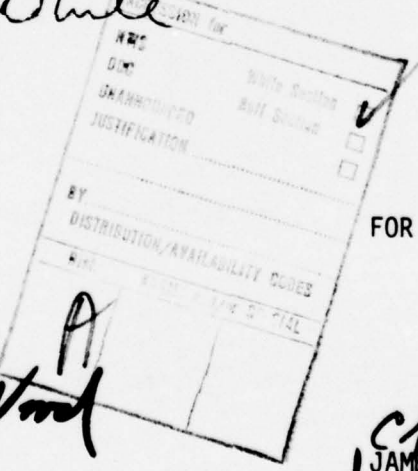
When US Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government therby incurs no responsiblility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, speicifications, or other data is not to be regarded by implication or otherwise as in any manner locensing the holder or any other person or corporation or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related therto.

This technical report has been reviewed and is approved for publication.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

A. Brent White

A. BRENT WHITE
Project Officer



FOR THE COMMANDER

Larry W. Wood

LARRY W. WOOD
Lt Colonel, USAF
Chief, Phenomenology/Technology Branch

Charles E. O'Hara

JAMES L. GRIGGS, JR.
Colonel, USAF
Chief, Electronics Division

DO NOT RETURN THIS COPY. RETAIN OR DESTROY.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFWL-TR-75-159	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): DATA BASE MANAGEMENT SYSTEM REFERENCE MANUAL	5. TYPE OF REPORT & PERIOD COVERED: Final Report.
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s): James M. Phelan	8. CONTRACT OR GRANT NUMBER(s): F29601-74-C-0017
9. PERFORMING ORGANIZATION NAME AND ADDRESS: BDM Corporation, 6070 Gateway East Boulevard, Suite 411, El Paso, TX 79905	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62704H, WDNE1308
11. CONTROLLING OFFICE NAME AND ADDRESS: Director, Defense Nuclear Agency, Washington, D.C. 20305	12. REPORT DATE: October 1976
	13. NUMBER OF PAGES: 142
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Air Force Weapons Laboratory (ELP), Kirtland Air Force Base, NM 87117	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

This research was sponsored by the Defense Nuclear Agency under Subtask Z99QAXTC022, Work Unit 52, Work Unit Title: "Interface Program for Circuit Analysis".

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Analysis
Computer Aided Analysis
Data Management
Data Base Systems
CADA
Network Simulation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Data Base Management System (DBMS) is a computer program which creates and maintains a hierarchical set of data bases and data base entries in which both the data base structure and the data itself may be defined and controlled by the user. Information stored in the data base is stored as entity attributes. These entities may be placed into ordered sets and inverted linked lists as well as being organized along the normal hierarchical lines of the data base. Information may be retrieved to core storage, system files, and output devices.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

391886

CONTENTS

ILLUSTRATIONS

TABLES

SECTION I

INTRODUCTION

This report discusses the Data Base Management System (DBMS) which has been developed as a component program for a systems analysis program to be used in the vulnerability assessment of weapons systems to nuclear radiation effects. The DBMS has been developed along very general lines so that it is not oriented in any way towards the systems analysis application. The DBMS is organized as a hierarchical data base and supports ordered sets and linked lists. Several kinds of data types are supported and information may be retrieved to core storage, system files, and output devices.

The DBMS is able to operate as a stand alone program, or, with a suitable executive controller, in concert with other programs. Input to the DBMS may be provided in a user language which specifies basic DBMS operations, or through a list of bit-packed computer words. The bit-packed words are available for inter-program communication and accomplish the same operations as the commands in the user language.

Since the DBMS operates on basic commands it is convenient to have a higher level user language through which the user can issue macro commands to the DBMS. These macro commands are composed of other more primitive macro commands and basic DBMS commands. Appendix E discusses the concept of a Data Base Macro Processor (DBMP) which would implement such a feature. Figure 1 illustrates the data and command communication paths which are used by the DBMS and the proposed DBMP.

The DBMS performs the bookkeeping function in a data base. These bookkeeping functions are divided into four categories: functions which perform general utility tasks, functions which create and maintain the hierarchy of a data base, functions which control the fine structural detail of a data base as well as enter and retrieve data from the data base, and functions which control the operation of the DBMS. Section II describes all of the DBMS commands in each of the four categories. Section II also describes the general structure of a data base as created by DBMS. The details of how

the information is stored in the data base and the format of commands to DBMS are discussed in Section III. The primary algorithms used in the DBMS are presented in Section IV.

Sections II, III, and IV are organized as structure levels, each successive structure level presenting a more detailed discussion of the DBMS. Therefore, the reader is advised to read as many structure levels as necessary to understand, in sufficient detail, how to accomplish his current goals.

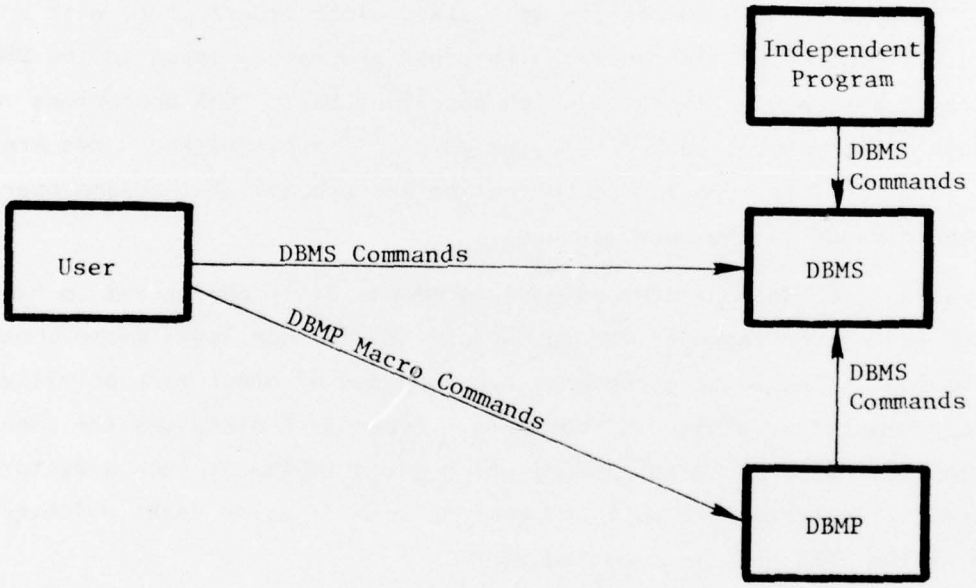


Figure 1. Command Communication Paths for DBMS.

SECTION II

BASIC DATA BASE STRUCTURE

STRUCTURE LEVEL 1.0: OVERVIEW OF DBMS

The basic structure of the DBMS will support multiple data bases. Each data base will support an unlimited number of data cells ordered in a tree of any level. Although the cells of any level need not contain the same type of data, all the cells on any level are arranged into groups of like structure (same type and number of data fields called attributes). Figure 2 is a generalized picture of a data base demonstrating the tree structure and

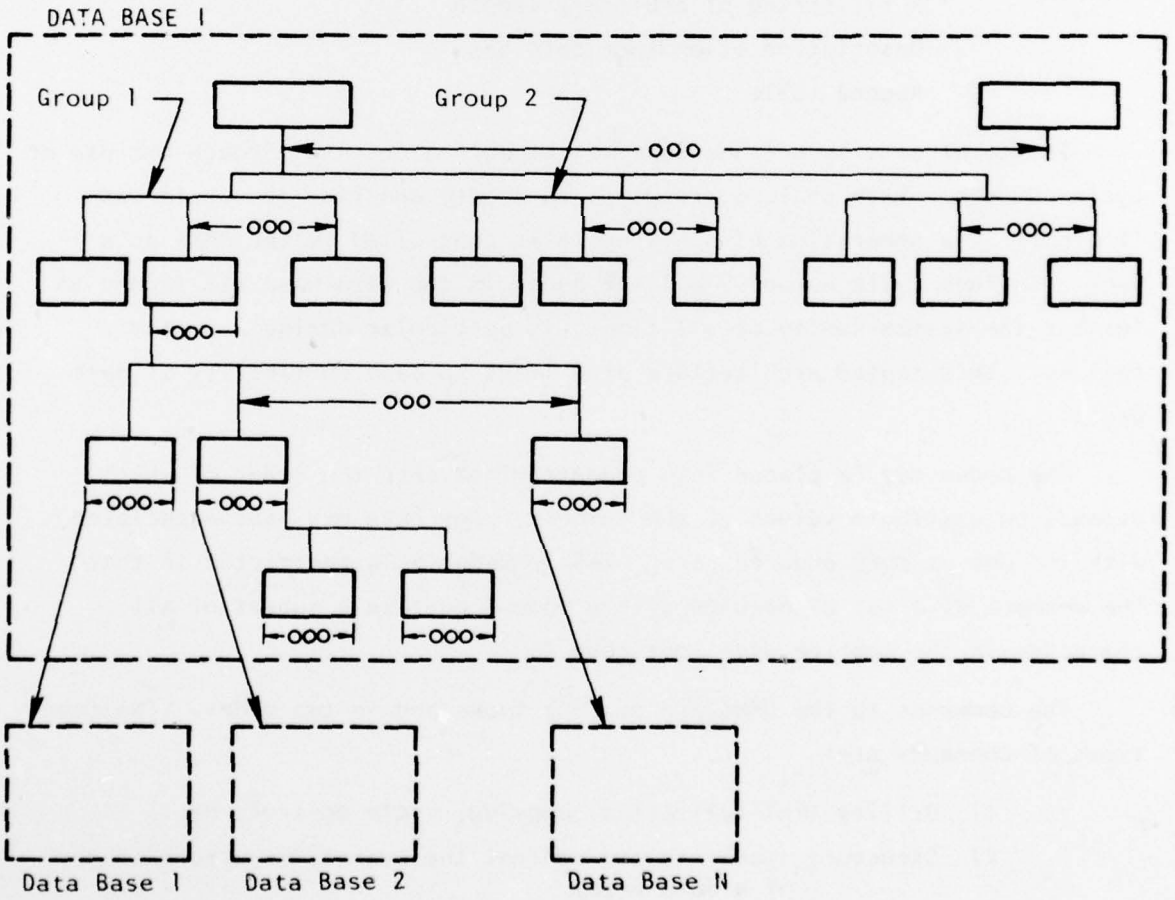


Figure 2. Generalized View of a Data Base

the generality obtainable with DBMS. Data may be stored at any level in the data base in the form of attribute values. An attribute consists of a name and a value which define a state of an entity. The attribute name may be from 1 to 9 characters and the attribute values may be stored as single variables, as simple arrays or as ragged tables (see Structure Level 1.2). A variable may be stored as any of the following:

Alpha string of from 1 to 649 characters
Real
Integer
Complex
Double precision
A bit string of arbitrary length
Description of another data base
Ragged table

The total data base is designed to be self archiving through the use of cycle numbers. Each cycle corresponds to a date and time the cycle was 'frozen'. The generation of a new cycle is controlled by the DBMP or a user. Besides cycle numbers, all the nodes in the data base are tagged to further the system sanity at all times, in particular during a system failure. This tagged architecture also leads to easy portability of data bases.

The nodes may be placed into predetermined sets the order of which depends on attribute values of the members. Any node may have associated with it, one or more ordered sets. Set membership is restricted in that the members of a set associated with a node A must be a subset of all the nodes in the subtree with root node A.

The commands to the DBMS are of four types and in two modes. The four types of commands are:

1) Utility (initialization, copying, cycle control, etc.)
2) Structure (generate and control the overall structure/shape of a data base)
3) Data (enter data into data base)
4) DBMS Control

The two modes consist of pure alphanumeric commands for human use and short commands for use by interprogram communications.

A data base is intended for primary storage on a random access device during creation, modification and referencing. The data base is readily stored on tape and may be easily transferred from tape to a random access device of a different brand computer. During modification, the DBMS references the data base through a paging algorithm.

Structure level 1 discusses the basic structure of the data base as a tree and the generation and control of a data base. In particular, the commands and their usage are discussed along with the data base tree and grouping of data items into sets and inverted linked lists. This discussion is intended as an introduction. The casual user of DBMS may, however, find this introduction adequate.

STRUCTURE LEVEL 1.1: BASIC COMMANDS TO THE DBMS

The commands to the DBMS are of four types: utility, data base structuring, data entry/retrieval, and DBMS control. A general description of the commands and their effects are discussed here. Subsequent structure levels contain more details, and Appendix A contains command syntax (note commands are listed in a logical order here, but are listed alphabetically by type in Appendix A).

1) Utility Functions:

   A) INITIALize a data base - performs file opening and establishes the basic structure of a data base.

   B) RETRIEVE an existing data base - validates permission keys to prevent accidental destruction of an existing data base.

   C) COPY a data base from random access device to random access device or to tape. Also allows deletion of superfluous cycles of a data base.

   D) RESTORE a data base from tape to random access. The tape may have been generated on a different brand of computer with compatible tape transports.

   E) LIST out date and time of all cyc'es currently archived in the data base.

F) DISPLAY structure and/or attribute values of all or part of the current or previous cycles of a data base.

G) COMPRESS a data base on a random access device by removing wasted space caused by fragmentation.

H) Increment the current CYCLE number by one, thus archiving a copy of the data base.

2) Structure Functions:

A) Add a STRUCTURE group to the current data base. This command is used to describe the characteristics of a group (e.g. Group 1 of Figure 2). The information required is: the groups of nodes which are siblings; the groups of nodes which may belong to sets of this group; and the attributes of the members of this group.

B) LINK/UNLINK a group of nodes into an ordered list.

3) Data insertion, modification and retrieval functions:

A) CREATE a new data node

B) ENTER a data node into a set.

C) DELETE and/or RETURN a data node and its attributes.

D) STORE an attribute value of a DBE.

E) FIND a data node.

4) DBMS Control

A) END of DATA

B) Change MODE of input

Upon examining functions of Type 2 (structure functions) and Type 3 (data functions) it can be seen that the data base is intuitively divided into two sections. One half contains general structure information and the second half contains stored data in a compacted form and specific details of structure, namely set membership and ordering within sets and lists. Both sets and lists will be discussed in subsequent structure levels, but the following comparison of the two will be useful now. Both sets and lists

have data nodes as members, and both may be ordered according to a queueing discipline or by ranking of attribute values. A set contains only data nodes which have been entered thru an ENTER function. An ordered list contains all data nodes on a subtree which satisfy a specific attribute relational expression. This distinction is important, since both concepts are very useful, as will become evident in subsequent structure levels.

STRUCTURE LEVEL 1.2: DATA BASE STRUCTURE

For every group of data nodes (e.g., Group 1, Figure 2) there corresponds one and only one structure node (SN), hence SN will be used henceforth in place of group. The structure nodes form a tree from which the data base derives its hierarchical structure. Each structure node is given an alpha-numeric name consisting of from one to ten characters starting with any letter. The root of the SN tree is always named 'SYS'. The SN called 'SYS' is primarily generated and manipulated by the DBMS and the DBMP (see level 3, Data Base Maintenance). There is only one restriction on the name of a SN: the names of SNs within a structure set and within the siblings of any node must be unique. The meaning of the term 'structure set' will be defined below; for now a structure set can be considered as a user defined subset of SNs on any subtree of the data base tree. Duplications are allowed anywhere else. Any SN can be uniquely defined by its generic name (the names of all its ancestors separated by periods). In Figure 3a the leftmost bottom node is 'SYS.A.B' and the right most bottom node is 'SYS.A.A.B'. Nodes 'SYS.A' and 'SYS.A.A' cannot be in the same structure set since their names are both 'A'. Similarly the nodes 'SYS.A.B' and 'SYS.A.A.B' cannot be in the same structure set since they are both named 'B'. Figure 3b shows another naming error, in particular SN'A' has two siblings named 'B'. The naming of SNs in Figure 3c avoids the problems of Figure 3a and 3b. Note that the naming of SNs in Figure 3a will not cause problems as long as the structure sets are properly chosen.

With each SN there can be associated zero or more data base entries (DBE). It is the data base entries which contain the attribute values.

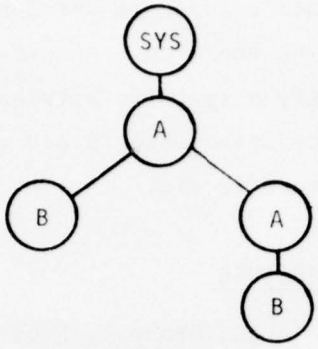


Figure 3a. Structure Nodes

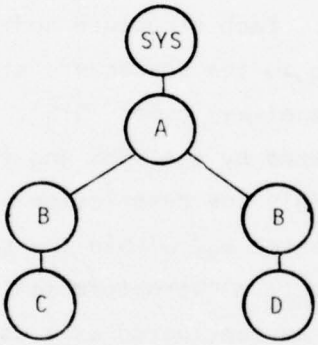


Figure 3b. Illegal Structure Nodes

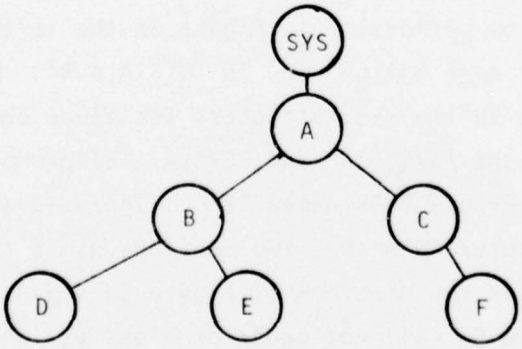


Figure 3c. Structure Nodes

Each SN acts as a template for its associated DBEs. Figure 4 shows an idealization of a data base limited to the structure previously discussed. Each SN is shown as the left column of a block and the DBEs associated with a SN are attached as columns to the right of the SN. The SN names are at the top of a SN column and attribute names are shown below the SN name. The attribute values are shown for each DBE (e.g. the first SN below 'SYS' is 'SIMS' which has two DBEs). The attribute values indicate two simulations, 'SIM1' and 'SIM2' which were originated by organizations 'A' and 'B' respectively. The next level of SNs corresponds to the type of devices (transistors and diodes) which are input to the simulations. Note that the two siblings of SN 'SYS.SIM' (named 'XTORS' and 'DIODE') have different attributes. All of the transistor models could be referenced by 'SYS.SIMS.XTORS'. The attribute DOMCODE is assumed to be '1' for frequency domain and '2' for time domain. By listing out (or searching through) all the attribute values for 'SYS.SIMS.XTOR.MODELS' a user can locate an existing transistor model which can be input to simulation 'SIM1' or 'SIM2'. Note that by using the structure presented so far some information had to be duplicated in the DBEs associated with SN 'SYS.SIM.XTORS.MODELS'. In particular the attributes 'SIMS' and 'DOMCODE' contain information corresponding to SN 'SYS.SIMS' and the attribute 'DOMCODE' in SN 'SYS.SIMS.XTORS'. Since there would probably be many transistor models in the data base, much space would be needlessly wasted. This duplication could have been avoided had 'SIM1' and 'SIM2' been SN's instead of 'SIMS', and 'DOM1' and 'DOM2' been SNs just after SN 'SYS.SIMS.XTORS', but then the listing out of all the transistor models would have been more difficult. The problem is simplified with the concept of sets and lists. For example all the transistor models associated with SN 'SYS.SIMS.XTORS' can be entered into one of four sets according to which simulation and domain is appropriate. Also the user has the ability to define a transistor model and not have it considered as input to either simulation until he is satisfied with the model's accuracy, at which time he enters it in the appropriate set. The concept of sets will be presented next; the concept of lists is discussed somewhat later.

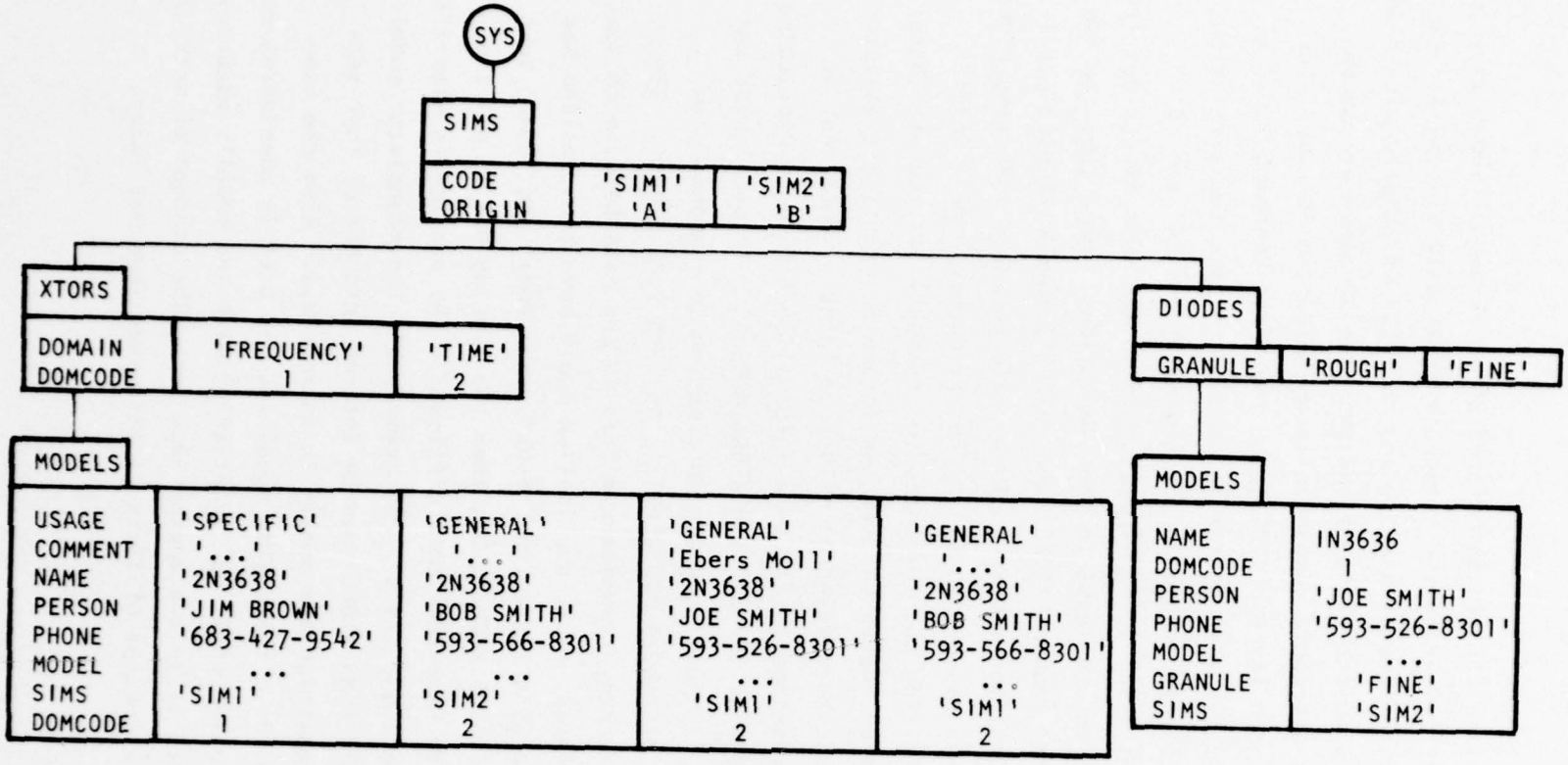


Figure 4. Basic Structure of a Data Base Without Sets or Lists

The level of any SN is the number of his ancestral nodes. Considering Figure 3b, the level of SN 'SYS' is 0, the level of SN 'SYS.A' is 1, the level of SN 'SYS.A.B' is 2, and the level of SN 'SYS.A.B.C.' is 3. The level of any DBE is the level of its associated SN. Considering Figure 4, all the DBEs for transistor models are of level 3.

Intuitively, a DBE set belongs to a DBE called the owner, and a DBE may own at most one DBE set. The members of a DBE set are specified individually by a user (see Type 3 function ENTER in structure level 1.1) and must consist of DBEs having a lower level than the DBE set owner. The DBE set members are said to belong to the DBE set owner. A DBE may belong to more than one DBE on higher levels, but must belong to no more than one DBE of a given level. A structure set of SN 'A' consists of nodes on the structure subtrees with the children of SN 'A' as roots. Similar to a DBE set, a SN may own at most one structure set. Structure set members must be of a lower level, and a SN may belong to at most one SN of a given level. The members of a structure set are specified when the structure set owner or member is described (Type 2 function STRUCTURE). A structure set defines potential members of a DBE set. DBE 'A1' with associated SN 'A' may only own DBEs whose associated SNs are in the structure set of SN 'A'. A structure set differs from a DBE set in the types of nodes which are members, and each SN owns all of its children by default. Figure 5 shows a group of SNs (squares) and DBEs (circles). Here SN 'A' may own a structure set of ('B') or ('B', 'C'), or ('B', 'D'), or ('B', 'C', 'D').

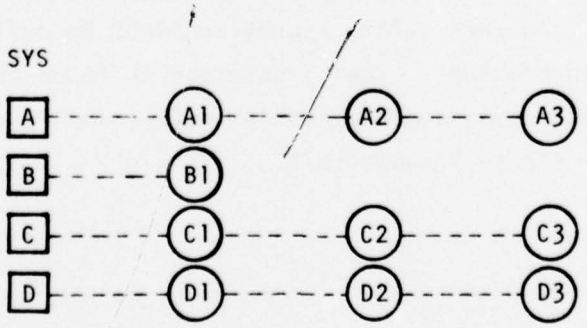


Figure 5. Data Base Entries

SN 'D' may not own a structure set. If SN 'A' has a structure set of ('B', 'D'), then DBE 'A1' may own a DBE set but DBE 'C1' may not belong. DBE 'D1' and/or 'D2' and/or 'D3' may belong to the DBE set owned by 'A1', but if DBE 'D1' belongs to the DBE set of 'A1' then 'D1' may not belong to the DBE set of 'A2' or 'A3'. In the future the phrase 'the DBE set belonging to the DBE A' will be shortened to 'the set of A'. The concept of set is in keeping with the general data base philosophy of keeping the DBEs small by placing as much structure information into SNs which then act as templates for the DBEs. The members of a set may be ordered by queueing discipline (FIFO or LIFO) and/or ranked by a numeric attribute value (increasing or decreasing). As such, the set information (who may belong to whom, etc) is passed to the DBMS as a structure set definition when a SN is defined (see Type 2 command STRUCTURE). The same is true for linked list definition which will be defined later.

Figure 6 is a copy of Figure 4 with set relations emphasized. For the sake of argument, assume SN 'SYS' owns SN 'SIMS' and 'XTORS', and further assume all other SNs besides SN 'SYS' only own their children. In Figure 6, the SNs are to the left of the double line and their associated DBEs are connected by a dashed line. Note that the SN 'SYS' has a corresponding DBE which may own any DBE associated with SN 'SIMS' or SN 'XTORS'. In Figure 6 SN 'SIMS' owns SN 'XTOR' and SN 'DIODES'. The first DBE of SN 'SIMS' (Code = 'SIM1') owns two DBEs of SN 'XTOR'. Now it is very easy to locate all transistor models or just those for simulation 'SIM1' in the time domain.

Another use of sets might be to simplify the bookkeeping of problems and corresponding solutions. In particular, a problem might be defined and placed into a set of unsolved problems. When a solution is found the solution is filed with the problem definition and the problem definition is moved to the set of solved problems. Figure 7 shows this.

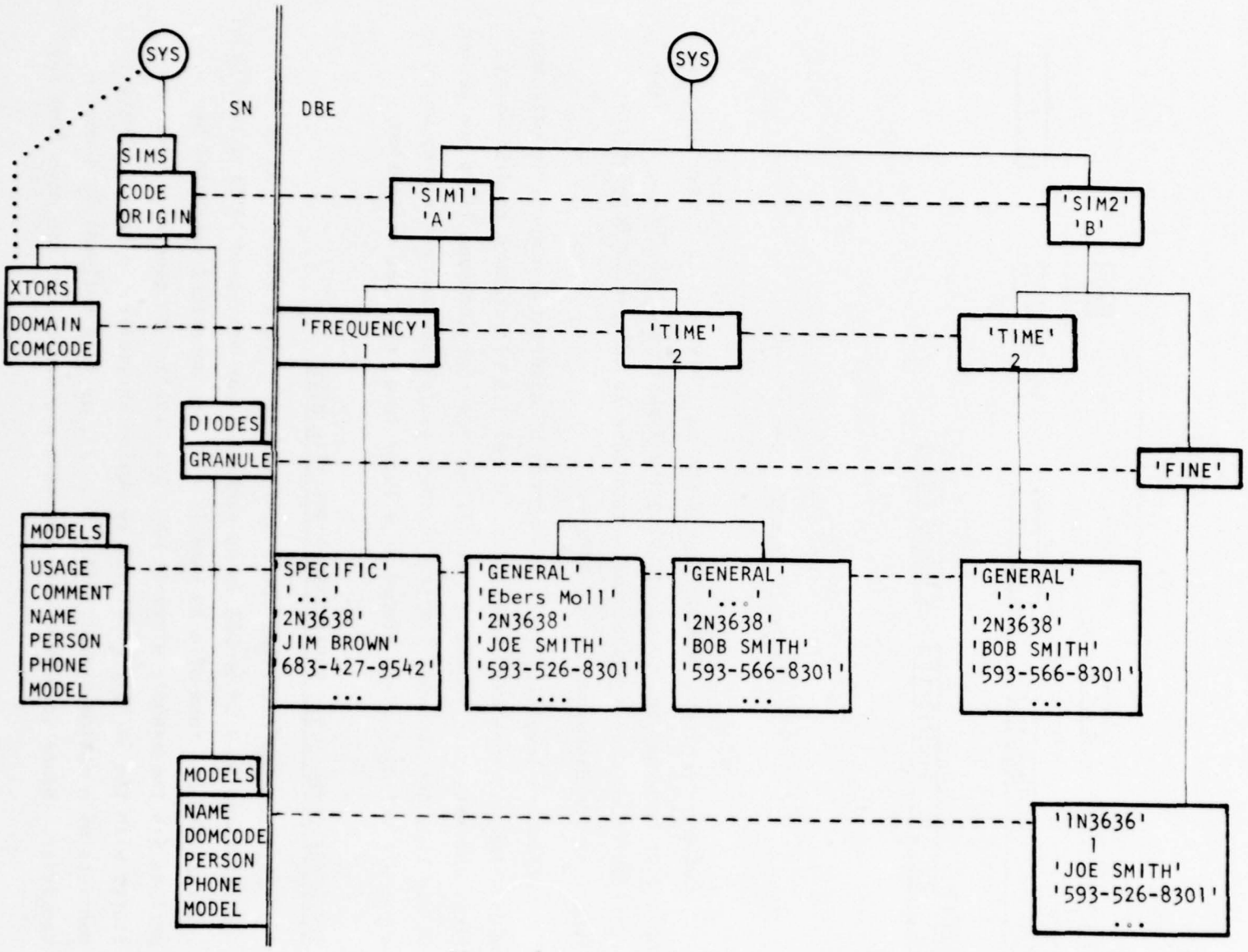


Figure 6. Sets and Lists

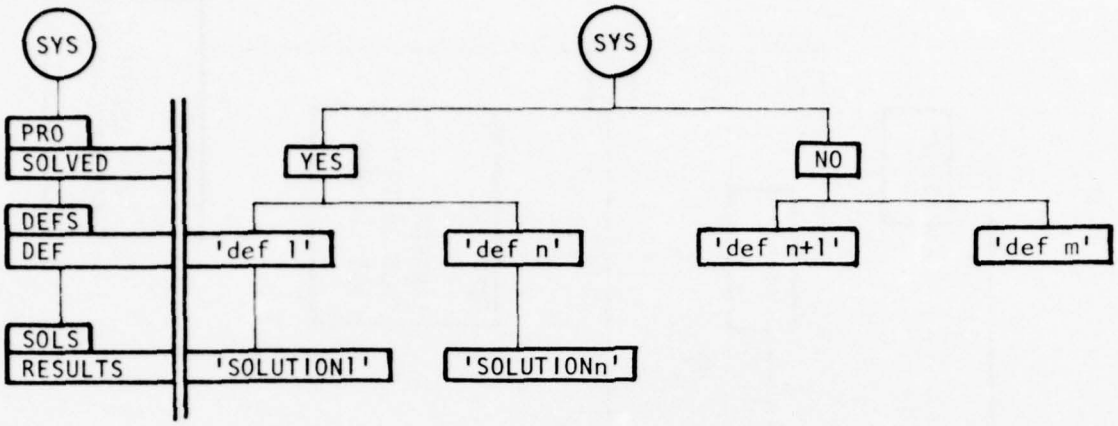


Figure 7. Another Use of Sets

Cycle information is kept on all SNs and DBEs. As such previous states of a data base may be recreated using the Type 1 function COPY. The Type 1 function CYCLE causes a state of a data base to be saved. Both of these functions are discussed in Appendix A.

Structure level 1.3 discusses a mode of describing a specific data base NODE which enhances the use of sets. Level 1.5 will return to discussing lists and sets. It should be pointed out now that although lists are easier to use they are somewhat limited in their usefulness and efficiency which is why their definition was placed at a later level than set definition.

STRUCTURE LEVEL 1.3: SPECIFYING A SPECIFIC NODE

This section describes how a user can refer to a specific SN or DBE. The reference to a SN or DBE is by name. As was mentioned in Structure Level 1.2 for SNs, one form of a SN name is the SN's ancestors separated by periods (in the example given it was 'SYS.A.B.'). In general a node name starts with the SN 'SYS' followed by motion phrase(s). A motion phrase consists of a motion symbol ('$', '*', '/', or '.') followed by a motion specifier. Since structure sets and sets are ordered we can move from any

node to the first or last member of the node's set, denoted '$F' and '$L' respectively. Also while in a DBE set or a STRUCTURE set we can move to the preceding or succeeding node, denoted '$P' and '$S' respectively. Attribute values may be used to specify a particular member of a DBE set; this is denoted by a dot followed by a relational expression.

In the DBMS a relational expression can have one of two forms. The first form consists of only the attribute name; it has a value of true if and only if the attribute name is present in the DBE being considered.

The second form consists of a standard ANSI FORTRAN logical expression. The relational operators .EQ., .NE., .GT., .GE., .LT., and .LE. may be used; however, when non numeric attribute types are involved only .EQ. and .NE. are permitted, and all operators in the expression must be of the same attribute value type. Logical expressions may utilize the logical operators .AND., .OR., and .NOT., and parentheses may be used for grouping purposes. The entire logical expression must be enclosed by parentheses.

Another motion is from a SN to its associated DBEs, which is denoted '*'. The symbol '*' is followed by '$F', '$L', or '/'. Whereas the '.' produced motion into a set or list, the '/' produces motion within a set or list (if '/' follows '*' the motion is within all DBEs associated with the SN being left). The symbol '$B' moves from a SN to its parent or from a DBE to its associated SN. As a final note before the examples, the ordering of DBEs was specified by their associated SNs (Type 2 function STRUCTURE which created the SN). As such, the ordering of DBEs determine which DBE is used when more than one DBE would satisfy an attribute relational specification.

The reader is referred to Figure 6 for the following examples. The following all specify the SN for transistor models:

1) SYS.SIMS.XTORS.MODELS
2) SYS.SIMS.XTORS$F
3) SYS.SIMSLP.MODELS

The following all specify the transistor model written by 'BOB SMITH' for simulation 'SIM1' (third from left):

1) SYS.SIMS.XTORS.MODELS*/(PERSON.EQ./BOB SMITH/)
2) SYS*/SYS.(CODE.EQ./SIM1/).(DOMCODE.EQ.2)$L
3) SYS.SIMS.XTOR*/(DOMAIN.EQ./TIME/)$L

If just any 2N3638 time domain transistor models is desired then we can use

SYS.SIMS.XTOR*/(DOMAIN.EQ./TIME/) . (NAME.EQ./2N3638/)

A short node naming technique is also possible. Type 2 functions can specify a SN name, and the DBMS remembers the current SN name. A SN name is considered current for the Type 2 command specifying it and all the commands prior to the next Type 2 command. Similarly for all Type 3 commands except CREATE, the DBMS keeps track of the current DBE. For the CREATE command the DBE created is considered the current DBE and the associated SN is considered the current SN. Thus the short form for specifying a SN field or DBE field starts with a motion phrase and may be followed by more motion phrases as needed. As an example, if in Figure 3 the current SN is 'SYS.A', then SN 'SYS.A.A' may be referenced in short form by '.A'.

Appendix B gives the result of motion phrases for SNs, DBEs, structure sets, DBE sets and linked lists. At this time it suffices to mention that motion in a linked list is similar to motion in a set.

STRUCTURE LEVEL 1.4: ATTRIBUTE VALUES

The form of storing an attribute value must be specified when the attribute is defined (See Appendix A, Type 2 function STRUCTURE). However, array dimensional information may also be defined (or redefined) when a DBE is created or altered.

The dimensionality of a static (or simple or FORTRAN) array is defined dynamically (Type 3 function SET) by stating the dimension(s). For array X, one could say X(*,*) = 3, 4 which would set the attribute X in the DBE to a three by four array.

A ragged table is more versatile. First of all, the information stored does not have to be of all the same kind, and the number of columns does not remain constant. Figure 8 is an example of a two dimensional ragged table. For more dimensions than three it is easier to show a ragged table as a tree,

10	12	83	36
4.8	5.6		
72			
98.	7.	4.5	.15
17			
22	3	38	

Figure 8. A Simple Ragged Table

as in Figure 9. Note the leaves of a ragged table (tree) do not need to be of the same level or of the same type (integer, etc). Although the number of nodes in a ragged table is limited to about 20000, the leaves of a ragged table can be any type of attribute value including other ragged tables. Thus a ragged table, as seen by a user, is of unlimited size.

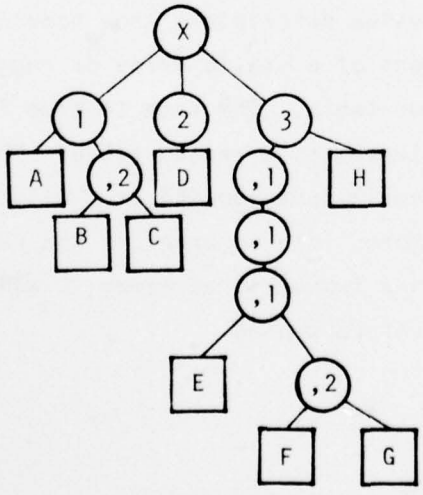


Figure 9. A Ragged Table Shown as a Tree

Similarly, although the maximum number of DBEs permitted in a data base is about 10^7, the DBMP can use a ragged table of data bases to create a data base of unlimited size.

Referring to Figures 8 and 9, some efficiency is gained if the leaves of a ragged table are static arrays. The specifying of dimensions in a ragged table is different from static arrays, in that the number of branches at any node must be defined. Thus the required definitions for the dimensions and attribute value type of the leaves of Figure 9 are as follows:

```
X(*) = 3
X(1,*) = 2
X(1, 1, *) = A
X(1, 2, *) = 2
X(1, 2, 1, *) = B
X(1, 2, 2, *) = C
X(2, *) = 1
X(2, 1, *) = D
X(3, *) = 2
X(3, 1, *) = 1
X(3, 1, 1, *) = 1
X(3, 1, 1, 1, *) = 2
X(3, 1, 1, 1, 1, *) = E
X(3, 1, 1, 1, 2, *) = 2
X(3, 1, 1, 1, 2, 1, *) = F
X(3, 1, 1, 1, 2, 2, *) = G
X(3, 2, *) = H
```

where A, B, C, D, E, F, G are attribute value descriptors (see Appendix A, Type 2 function STRUCTURE). The dimensions of a static array or ragged table are attribute values of the array or table. The same is true for the attribute value description for the leaves of a ragged table. Thus all three of the above may be set with the Type 3 function SET or the Type 2 function STRUCTURE. The variables are stored into an array in the FORTRAN sense. If A is a single integer and B is a two by three array, A and B(1,3) could be set by the Type 3 function SET referenced as

```
X(1, 1) = 100
X(1, 2, 1, 1, 3) = 101.
```

An attribute value entity of type alpha (or bit) string may be of any length up to about 30000 characters (or bits).

The dimensions of a static array or ragged table may be referenced in the relational expressions mentioned in Structure Level 1.3.

An application of ragged tables is the storage of sparse matrices.

STRUCTURE LEVEL 1.5: LINKED LISTS

Whereas a set is attached to (owned by) a DBE, a linked list (LL) is attached to a SN. The membership is defined by attribute names being present and/or attribute value relational phrases (See Structure Level 1.4). All DBEs which 1) meet the attribute requirements on names and values, and 2) which are associated with any SN which is a descendent of the linked list's SN, are automatically members of the list. The list is ordered by numerical attribute values and/or a queueing discipline. Further, list membership is independent of when a DBE is created. Ordering of a LL is specified by a series of ordering phrases. The first phrase is used for primary ordering. Subsequent phrases are used to break ties. Unless a 'LIFO' discipline is specified, the final ordering phrase is assumed to be a 'FIFO'. If a 'LIFO' or 'FIFO' discipline is used for DBEs present before the LL is created, the queueing discipline becomes cycle no., then level no., and then relative time of original association with a SN (See Appendix A, Type 2 function LIST and Type 3 function CREATE). Note when a DBE is created its attribute values are flagged as non existent until a value is entered. Finally, a SN may have zero or more lists.

As an example, Figure 10 shows a data base with a LL. The LL is called L1 and contains DBEs 'B1', 'C1', 'C2', 'B3', and 'B2'. Figure 11 shows the same LL after DBE 'B3' has been removed and DBE 'E3' has been added. Note that 'C3' fits between 'C1' and 'C2' in the list ranking. Note that DBE 'A1' is not a candidate for LL 'L1' and that SN 'SYS' may have LLs attached. Figure 12 shows the same data base with a second LL attached to SN 'A'. The second LL is called 'L2' and its members are linked together with a squiggly line.

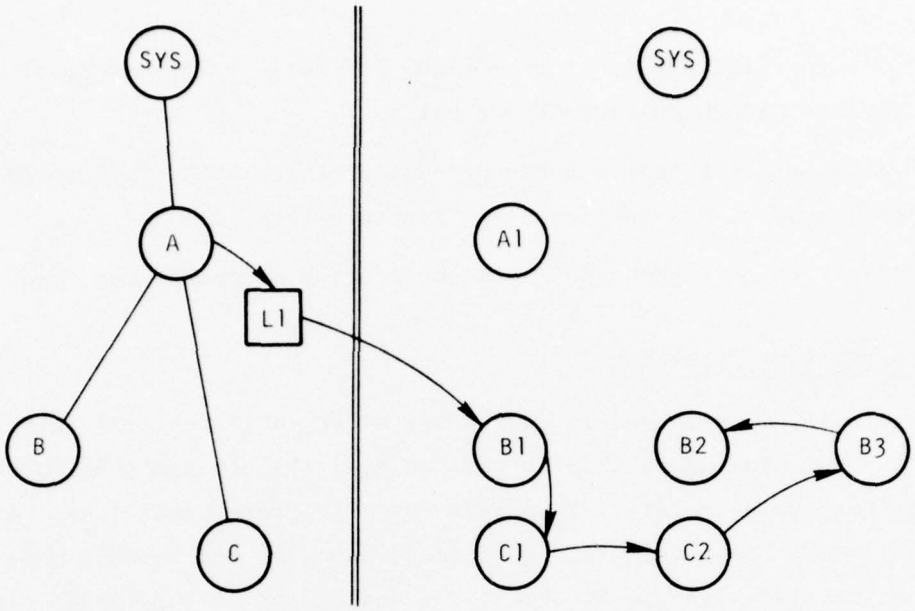


Figure 10. Linked Lists

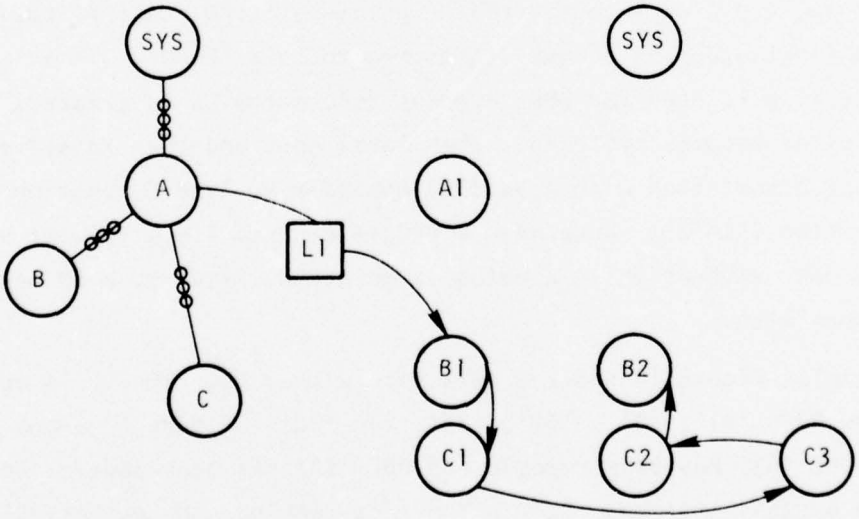


Figure 11. Linked Lists After Modifications

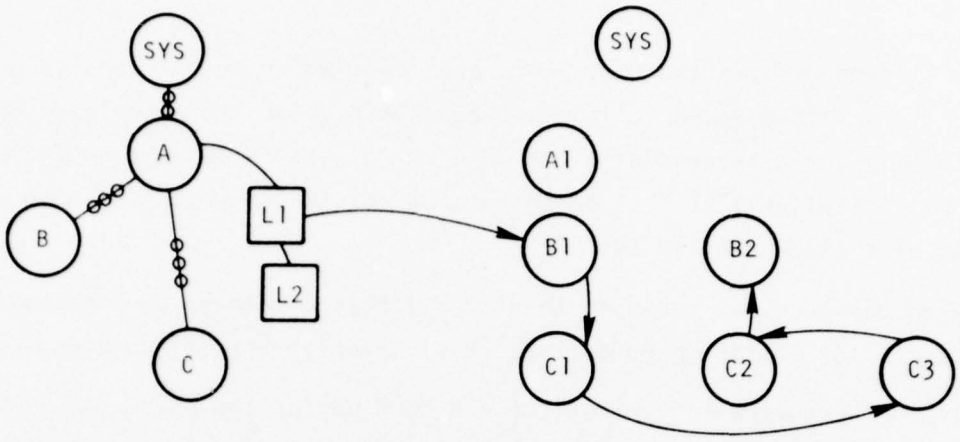


Figure 12. Multiple Linked Lists on a Structure Node

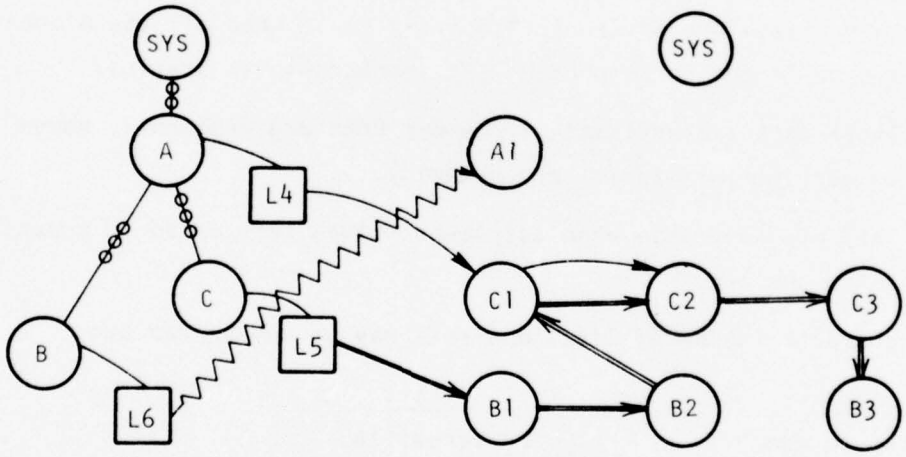


Figure 13. Linked Lists Which Will Never Occur

Figure 13 demonstrates two LLs, 'L4', and 'L5', which cannot legally occur. LL 'L5' is attached to SN 'C' and contains DBEs which are associated with a SN that is not a descendent of SN 'C' (SN 'B' and SN 'C' are on different branches). Similarly LL 'L6' contains a DBE of SN 'A' which is an ancestor instead of a descendent of SN 'B'.

When to use a set and when to use a list is dependent on the specific situation. The following guidelines should clarify most situations.

1) Sets require more effort of the DBMP and/or the user, but lists require more bookkeeping by the DBMS. Thus large lists should be avoided when a set will do. Lists are also expensive with respect to computer resource time when their range spans many SNs. It should be kept in mind that the DBMS is not assumed to be 'smart', whereas the DBMP is.

2) Large lists are advisable (contrary to 1) when all the elements would always be in one corresponding set (membership is static).

3) Small sets are advisable if member DBEs are frequently moved from set to set (membership is very dynamic).

4) Sets are advisable when attribute values referenced in ordering are frequently changed.

The characteristics of lists and sets may be summarized by:

lists	sets
easy to use	versatile
no limit to no. of lists	efficient
static	dynamic

SECTION III

INTERNAL DATA STRUCTURES USED BY DBMS

STRUCTURE LEVEL 2.0: DATA BASE BUILDING BLOCKS

Internally the data base is a tagged architecture. In particular, any entity which can have a pointer to it has a header describing the contents of the information contained in the entity. The header is called the tag and the information is called the text. The text may contain other nested entities which have tags and text. In the data base, the principal tagged entities are called nodes. The nodes may contain entities called cells, some of which are tagged. Structure Level 2 describes the internal representation of nodes as stored in the data base.

STRUCTURE LEVEL 2.1: CELLS

The smallest module of information storage in the data base is a cell. Table 1 lists the various cell types and formats for CDC 6600/7600 and IBM 360/370. Note that except for Types 2 and 7, the cell size remains constant from one machine to another with only the internal representation varying. This enhances the data base's portability. The cells have been defined primarily for the CDC 6600/7600 with consideration given to the IBM 360/370 and the UNIVAC 1108/1110.

STRUCTURE LEVEL 2.2: INTERNAL INPUT FORM

When the input mode is LONG the input formats are shown in Appendix A. DBMS converts the LONG input to a SHORT internal form. When the input mode is SHORT, DBMS uses the internal form without any conversion. The internal form consists of from one to seven arguments per DBMS command: command identifier, up to five keyword descriptions, and, if the input device is core, a flag indicating an argument has tried to reference a nonexistent node or nonexistent entry in a ragged table. The command identifier is a cell of

CELL TYPE	CELL CONTENTS	CDC STORAGE	IBM STORAGE
1	PNTR1 | PNTR2	PNTR1 | PNTR2	PNTR1 PNTR2
2	VAL1(32-60)	VAL1	VAL1
3	VAL1(6) | VAL2(12) | VAL3(12) | PNTR1	VAL1 | VAL2 | VAL3 | PNTR1	VAL1 | VAL2 | VAL3 PNTR2
4	VAL1(30) | PNTR1	VAL1 | PNTR1	VAL1 PNTR1
5	PNTR1 | VAL1(8)	VAL1 | PNTR1	VAL1 PNTR1
6	CHARS(51) | VAL1(8)	VAL1 | 5CHARS | 4CHARS	VAL1 | 4CHARS 5CHARS
7	PNTR1 | #BITS(12) | BITS(#BITS)	PNTR1 | #BITS BITS	PNTR1 #BITS BITS
8	VAL1(4) | . . . | VAL12(4) | VAL13(12)	VAL1 | . . . | VAL12 | VAL13	VAL1 | . . . | VAL8 VAL9 | . . . | VAL13
9	VAL1(15) | VAL2(15) | VAL3(15) | VAL4(15)	VAL1 | VAL2 | VAL3 | VAL4	VAL1 | VAL3 VAL2 | VAL4

WHERE:

PNTR = Pointer (30 bits)

VAL = Integer field (No. of bits required is indicated)

CHARS = 9 compressed BCD characters requiring two fields of 28 and 24 bits each.

BITS = Bit string (Number of bits is #Bits)

All data items are right adjusted in their assigned bit spaces.

Table 1. Cell Types

the following format:

CHECK	FLAG	#PARAMS.	COMMAND#

Cell Type 9

where:

CHECK = 15 bit low order check sum of cells for sanity check only

FLAG = 1 if an argument references a nonexistent node or nonexistent entry in a ragged table.

#PARAMS = the number of keyword fields present

COMMAND# = the identification number for the command. (See Table 2).

Number	Command	Number	Command
1	COMPRESS	11	UNLINK
2	COPY	12	CREATE
3	CYCLE	13	MODE
4	DISPLAY	14	ENTER
5	INITIAL	15	END DATA
6	LIST	16	FIND
7	RESTORE	17	STORE
8	RETRIEVE	18	RETURN
9	LINK	19	DELETE
10	STRUCTURE	20	REMOVE
		21	UNFILE

Table 2. DBMS Commands

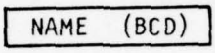
The keyword descriptions are tagged cells in the form of an array of one or more cells called a keyword packet. The first cell is Type 9 (See Table 1) and the first cell's rightmost field identifies which keyword phrase the array represents. The interpretation of the remaining information depends upon the keyword phrase. All the keywords (and thus keyword packets) fall into one of seven classes. Before showing the format of each case, some general comments about the keyword packets are in order. Keyword packets may be of an indefinite length. They may be written on a sequential FORTRAN file blocked with MPARRAY cells per logical record. The first four classes

of keyword packets (simple keywords, file definition, core definition, and linkage saving) require a few cells at most. The last three classes of keyword packets (expression, motion, and lists) are constructed of primary cells. These cells are presented before the keyword packets. Then formats of all seven keyword packets are presented, followed by a number of examples.

There are three classes of primary cells. They are given below, with the primary cell name listed first followed by the cell description.

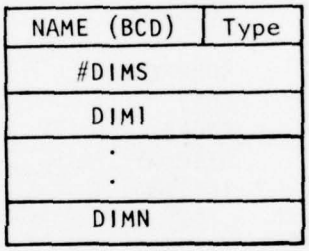
Class 1 Cells: Class 1 cells are simple primary cells. There are four types:

(1) N



Cell Type 6

(2) AN



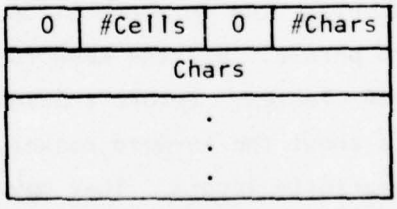
Cell Type 6
Cell Type 2
Cell Type 2

where Type = the attribute type (See Table 3) or 0. If not of type bits, array, or ragged table only the first cell is present.

#DIMS = number of dimensions

DIMi = the i^{th} dimension (for attribute type bits, 1st dimension is width)

(3) ACONST



Cell Type 9
Cell Type 2

where #Cells = the number of cells required for the alpha string plus 1.

#Chars = the number of characters in the string.

NUMBER	ATTRIBUTE TYPE
0	INTEGER
1	REAL
2	DOUBLE PRECISION
3	COMPLEX
4	ALPHA
5	BITS
6	DATA BASE DEFINITION
7	RAGGED TABLE

Notes: 1) Number +32 implies an array
2) 64 implies an attribute name

Table 3. Attribute Value Types

Chars = machine dependent character string must be an even number of words for IBM 360/370 or Univac 1108/1110.

(4) CONST

Text

Cell Type 2

where Text = a numeric constant

Class II Cells: There are five Class II cells. They are compound primary cells used in building keyword packets. They are:

(1) V1

AN
CONST

(2) V2

AN
AN

(3) V3

AN
ACONST

(4) V4

AN
LOC#

where LOC is a class 2 (file definition) or class 3 (core definition) attribute keyword packet (See below).

(5) V5

AN
DBDESC

where DBDESC is composed of 5 ACONS fields (in order they represent data base name, systkey, rkey, wkey, and akey)

Class III Cells: There are two Class III cells. These are complex primary cells representing a list of variables:

(1) VL

up to 12 SUBMODES	#Cells/seg
up to 12 Cells of Type V1, V2, V3, V4 or V5	
. . .	

Cell Type 8

where SUBMODES = 1, 2, 3, 4 or 5 for V1, V2, V3, V4 or V5 respectively

#Cells/seg = number of cells required for this VL (not including header).

(2) OAN

up to 12 SUBMODES	#Cells/seg
up to 12 cells of Type AN	
. . .	

Cell Type 8

where SUBMODES = 1, 2, 3 or 4 for HIGH, LOW, LIFO and FIFO

#Cells/seg = number of cells required for this OAN (not including header).

The seven classes of keyword packets (or keyword classes for short) are:

(1) Simple Keywords (CARD, DELETE, LISTS, LONG, PRINT, SETS and SHORT).

1	0	0	KEY #

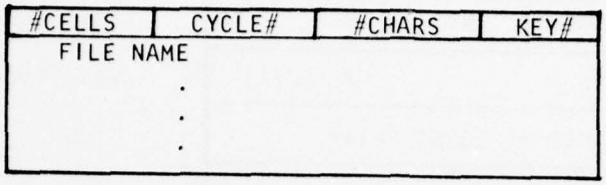
Cell Type 9

where KEY # = the identification number of the keyword phrase (See Table 4 below).

NUMBER	KEYWORD	NUMBER	KEYWORD
1	ABOVE	33	VALUE
2	ALL	34	(NOT DEFINED)
3	ATTRIBUTE	35	(NOT DEFINED)
4	BELOW	36	(NOT DEFINED)
5	CARD	37	(NOT DEFINED)
6	CLASS	38	(NOT DEFINED)
7	CORE	39	(NOT DEFINED)
8	DBE	40	(NOT DEFINED)
9	DBN	41	(NOT DEFINED)
10	DELETE	42	(NOT DEFINED)
11	EXCLUDE	43	(NOT DEFINED)
12	FILE	44	AFTER
13	FROM	45	ALPHA
14	INCLUDE	46	BEFORE
15	KEY	47	BITS
16	LINKS	48	COMPLEX
17	LISTS	49	DOUBLE
18	LL	50	DB
19	LONG	51	DELETE
20	MEMBER	52	FIFO
21	NAMES	53	FIRST
22	ORDER	54	HIGH
23	OWN	55	INTEGER
24	PRINT	56	LAST
25	RANK	57	LOW
26	SET	58	LIFO
27	SETS	59	RAGGED
28	SHORT	60	REAL
29	SN	61	SAVE
30	SYSKEY	62	$CORE
31	TAPE	63	$FILE
32	TO		

Table 4. DBMS Keywords

(2) File Definition (DBN, FILE, FROM, TAPE and TO):



Cell Type 9
Cell Type 2

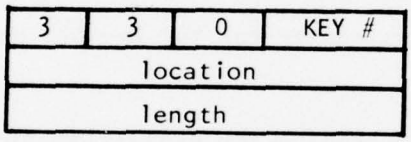
where #CELLS = the number of cells required for the file name + 1

KEY# = the identification # for the keyword phrase
(See Table 4)

CYCLE# = data base cycle number if specified (-1 is default)

#CHARS = the number of characters in the file name

(3) Core Definition (CORE):



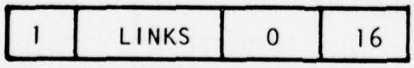
Cell Type 9
Cell Type 2
Cell Type 2

where KEY# = the identification number for the keyword phrase
(See Table 4)

LOCATION = an integer specifying starting location of commands

LENGTH = the number of cells in the command string.

(4) Linkage Saving (LINKS):



Cell Type 9

where LINKS = 1 if linkage is to be retained and 0 otherwise

16 = the identification number for the keyword LINKS

(5) Single expression (CLASS and SYSKEY)

#CELLS	MPARRAY	UNIT	KEY #	Cell Type 9
up to 12 SUBMODES			#SUBCELL	Cell Type 8
up to 12 AN, ACONST or CONST Cells				
. (Above two blocks repeated . as many times as necessary)				

where #CELLS = the number of cells required for the expression

MPARRAY = file blocking for keyword array (also number of words in core - 1)

UNIT = unit number for FORTRAN binary file containing the remainder of the keyword packet.

KEY# = the identification number for the keyword phrase (See Table 4)

SUBMODES = 4 bit tags describing the expression (See Table 5). Note the expression represented in Polish notation.

#SUBCELLS = total number of AN, ACONST or CONST cells

(6) Single motion (ABOVE, ALL, BELOW, DBE, LL, ORDER, SET, and SN):

#CELLS	MPARRAY	UNIT	KEY #	Cell Type 9
up to 12 SUBMODES			#SUBCELLS	Cell Type 8
up to 12 N or EXP cells				
. (Above two blocks repeated . as many times as necessary)				

NUMBER	(SUBMODE)/KEYWORD	NUMBER	(SUBMODE)/KEYWORD
0	null	9	.GT.
1	(Alpha)	10	.GE.
2	(INTEGER)	11	.OR.
3	(REAL)	12	.AND.
4	(AN)	13	.NOT.
5	.LT.	14	(BIT)
6	.LE.	15	Escape Code - See Note
7	.EQ.		
8	.NE.		

Note: Submode is (COMPLEX) if next numerical code = 1; submode is (DOUBLE PRECISION) if next numerical code = 2.

Table 5. Logical Expression Value and Operator Codes

where #CELLS = the number of cells required for the motion phrase

MPARRAY = file blocking for keyword array (also number of words in core - 1).

UNIT = unit number for FORTRAN binary file containing the remainder of the keyword packet.

KEY# = the identification number for the keyword phrase (See Table 4)

SUBMODES = 4 bit Tags describing the motion (See Table 6)

#SUBCELLS = Total number of N or EXP cells (Note that EXP cells are class 5 packets).

(7) Lists (ATTRIBUTE, EXCLUDE, INCLUDE, KEY, MEMBER, NAMES, OWN, RANK and VALUE).

#CELLS	MPARRAY	UNIT	KEY#	Cell Type 9
0	MODE	#LIST	0	Cell Type 9
LIST ELEMENTS				

where #CELLS = the total number of cells required by all the LIST ELEMENTS

MPARRAY = file blocking for keyword array (also number of words in core - 1).

UNIT = unit number for FORTRAN binary file containing the remainder of the keyword packet.

KEY# = the identification number for the keyword phrase (See Table 4)

MODE = basic mode of the list elements:

1 - AN
2 - ACONST
3 - M (single motion)
4 - VL
5 - OAN

NUMBER	(SUBMODE)/KEYWORD	NUMBER	(SUBMODE)/KEYWORD
0	null	8	*
1	$F	9	/
2	$L	10	(N)
3	$H	11	(EXP)
4	$S	12	FIRST
5	$P	13	LAST
6	$B	14	BEFORE
7	.	15	AFTER

Table 6. Motion and Ordering Codes

#LIST = number of elements in the list

LIST ELEMENTS = primary cells of Type N, VL or OAN,
or class 6 packets for type M.

A class 6 keyword packet may contain many class 5 keyword packets. Similarly a class 7 keyword packet may contain many class 6 and/or class 5 keyword packets. As is the case for all packets, contained packets may specify a unit (FORTRAN binary file) which contains the text of the packet. Since the header of a packet is easier to complete after the text is completed, the unit specified for a contained packet should be different from the unit of the parent packet. It is expected that up to three units may be required for a class 7 packet (1 for the class 7 packet text, 1 for all the class 6 texts, and 1 for all the class 5 texts). Since keyword packets are not always specified by the user in the order DBMS uses them, the units required for different keyword packets must be distinct. The Type 2 function, STRUCTURE, requires the most units; in particular, 10 units may be required (2 units for the keyword SN, 1 unit each for the keywords ATTRIBUTE and RANK, and 3 units each for the keywords MEMBER and OWN.

The following examples will illustrate each of the specific constructs. (The octal codes in the name fields of Type 6 cells are BCD character representations).

Example 1.

The keyword Packet for the keyword phrase

LISTS

is:

1	0	0	17

Cell Type 9, Keyword Class 1

Example 2.

The keyword packet for the keyword phrase

FROM = DOG

is:

2	-1	3	13
04 17 07			('DOG')

Cell Type 9, Keyword Class 2
Cell Type 2

Example 3.

The keyword packet for the keyword phrase

FROM = CATS(6)

is:

2	6	4	13
03 01 24 23 ('CATS')			

Cell Type 9, Keyword Class 2
Cell Type 2

Example 4.

The keyword packet for the keyword phrase

CORE = 1261 (25)

is:

3	2	0	7
		1261	
		25	

Cell Type 9, Keyword Class 3
Cell Type 2
Cell Type 2

Example 5.

The keyword packet for the keyword phrase

LINKS = SAVE

is:

1	1	0	16

Cell Type 9, Keyword Class 4

Example 6.

The keyword packet for the keyword phrase

CLASS = (LNGTH.GT.100 .AND. .NOT. WDTH.EQ.10)

is:

6			5			0			6
4	2	9	4	2	7	13	12	0--0	4
14	16	07	24	10		('LNGTH')			0
									100
27	04	24	10			('WDTH')			0
									10

Cell Type 9, Keyword Class 5
Cell Type 8
Cell Type 6
Cell Type 2
Cell Type 6
Cell Type 2

Example 7.

The keyword packet for the keyword phrase

SYSKEY = /TK=120396854//ID=GOOD/

is:

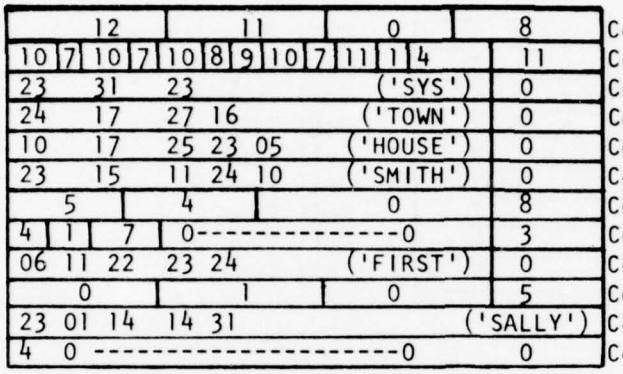
5			4			0			30		Cell Type 9
1	0----------							-0	3		Cell Type 8
0			2			0			20		Cell Type 9
24	13	54	34	35	33	36	44	41	43	('TK = 1203968')	Cell Type 2
40	37	50	11	04	54	07	17	17	04	('54/ID = GOOD')	Cell Type 2

Example 8.

The keyword packet for the keyword phrase

DBE = SYS.TOWN.HOUSE*/SMITH.(FIRST.EQ./SALLY/)FS$S

is:



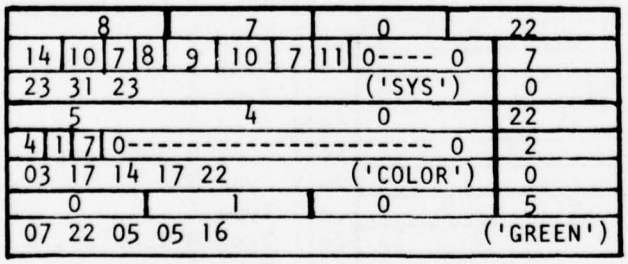
Cell Type 9 Keyword Class 6
Cell Type 8
Cell Type 6
Cell Type 6
Cell Type 6
Cell Type 6
Cell Type 9 Subclass 5
Cell Type 8 .
Cell Type 6 .
Cell Type 9 .
Cell Type 2 .
Cell Type 8

Example 9.

The keyword packet for the keyword phrase

ORDER = BEFORE SYS.*/SYS . (COLOR .EQ./GREEN/)

is:



Cell Type 9 Keyword Class 6
Cell Type 8
Cell Type 6
Cell Type 9 Subclass 5
Cell Type 8 .
Cell Type 6 .
Cell Type 9 .
Cell Type 2 .

Example 10.

The keyword packet for the keyword phrase

ATTRIBUTE = (ALPHA(DOG, CAT), INTEGER HOUSE, REAL PRICE(4,5))

is:

9	8	0	3	Cell Type 9 Keyword Class 7
0	1	4	0	Cell Type 9
04 17 07		('DOG')	4	Cell Type 6
03 01 24		('CAT')	4	Cell Type 6
10 17 25 23 05		('HOUSE')	0	Cell Type 6
20 22 11 03 05		('PRICE')	33	Cell Type 6
			2	Cell Type 2
			4	Cell Type 2
			5	Cell Type 2

Example 11.

The keyword packet for the keyword phrase

OWN = (SYS.STATE.CITY,.DIST,.SCHOOL,$B.HOSPITAL)

is:

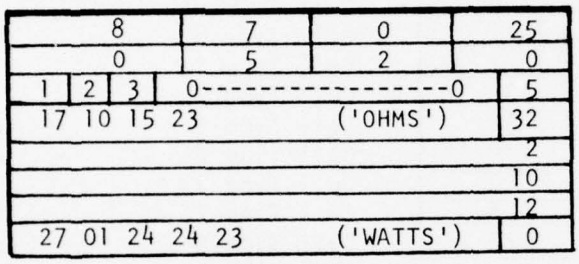
16	15	0	23	Cell Type 9 Keyword Class 7
0	3	4	0	Cell Type 9
5	4	0	23	Cell Type 9 Subclass 6
10 7 10 7 10	0----------0		3	Cell Type 8 .
23 31 23		('SYS')	0	Cell Type 6 .
23 24 01 24 05		('STATE')	0	Cell Type 6 .
03 11 24 31		('CITY')	0	Cell Type 6 .
3	2	0	23	Cell Type 9 Subclass 6
7 10 0----------0			1	Cell Type 8 .
04 11 23 24		('DIST')	0	Cell Type 6 .
6	5	0	23	Cell Type 9 Subclass 6
7 10 0----------0			1	Cell Type 8 .
23 03 10 17 17 14		('SCHOOL')	0	Cell Type 6 .
3	3		23	Cell Type 9 .
7 10	0----------0		1	Cell Type 8 .
10 17 23 20 11 24 01 14		('HOSPITAL')	0	Cell Type 6 .

Example 12.

The keyword packet for the keyword phrase

RANK = (HIGH OHMS (10,12) LOW WATTS LIFO)

is:



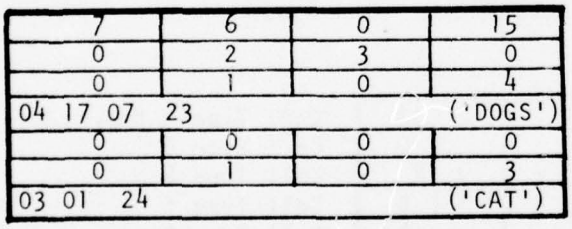
Cell Type 9 Keyword Class 7
Cell Type 9
Cell Type 8
Cell Type 6
Cell Type 2
Cell Type 2
Cell Type 2
Cell Type 6

Example 13.

The keyword packet for the keyword phrase

KEY = DOGS//CAT

is:



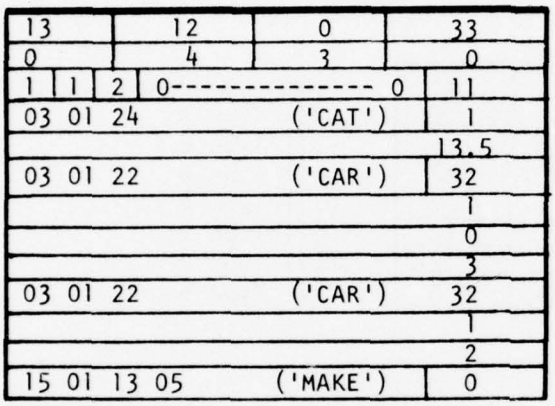
Cell Type 9 Keyword Class 7
Cell Type 9
Cell Type 9
Cell Type 2
Cell Type 9
Cell Type 9
Cell Type 2

Example 14.

The keyword packet for the keyword phrase

VALUE = (CAT = 13.5, CAR(*) = 3, CAR(2) = MAKE)

is:



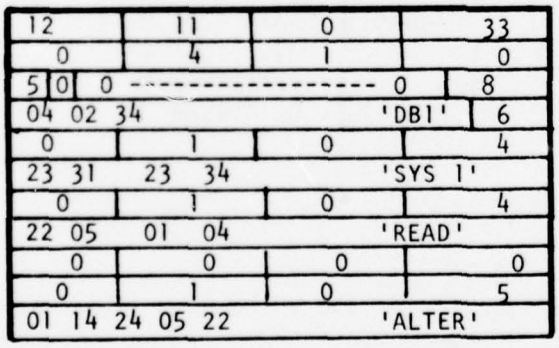
Cell Type 9 Keyword Class 7
Cell Type 9
Cell Type 8
Cell Type 9
Cell Type 2
Cell Type 9
Cell Type 2
Cell Type 2
Cell Type 2
Cell Type 9
Cell Type 2
Cell Type 2
Cell Type 9

Example 15.

The keyword packet for the keyword phrase

VALUE = (DB1 = *SYS1*$READ//ALTER)

is:



Cell Type 9 Keyword Class 7
Cell Type 9
Cell Type 8
Cell Type 6
Cell Type 9
Cell Type 2
Cell Type 9
Cell Type 2
Cell Type 9
Cell Type 9
Cell Type 2

When the input medium is core, the flag indicating an illegal node reference is a single cell of Type 4 containing a pointer to the beginning of the offending motion description (Keyword Class 6 array) and the number of successful motion phrases. Similarly, if an illegal ragged table index is detected, the flag will be a single Type 4 cell pointing to the beginning of the offending attribute reference and the number of the index which caused the fault.

As mentioned a command may be submitted to DBMS in a long or short mode from cards, from a file or from a core array. The format used in conjunction with the long mode from cards is detailed in Appendix A. The format used in conjunction with the long mode from a file or core array is simply 80 column card images. It is assumed the tape was created with a formatted FORTRAN write which creates a machine dependent tape (the format is 7A10, A2 for CDC, 18A4 for IBM, and 12A6 for UNIVAC or equivalent).

The format of the short mode strongly resembles the internal form just discussed but is dependent upon the input medium. When the input medium is a core array the command identifier, all five keyword packets (a header of zero is a null keyword) and the illegal node flag are located sequentially in core. The user is required to set proper values into the fields MPARRAY and UNIT. Thus the examples 1, 2 and 6 of keyword packets just presented would have octal representation in CDC core as:

Example 1a:

```
0000100000000000021
```

Example 2a:

```
0000300002000000015
0000077777000030000
0417075555555555555
```

Example 6a:

```
00006000050000000006
20512047670000000004
00037076131023306635
00000000000000000144
00072347265523306635
00000000000000000012
```

When the input medium is cards or a file, the command identifier and all five keyword packets appear sequentially as in the case for core, but portions of large keyword packets cannot reside on auxilary files. For card input or file input, the keyword packets must be complete and DBMS may place portions of large arrays onto files and set MPARRAY and UNIT in the tag of the keyword packet to appropriate values. If a file is the input medium, it is assumed to be a FORTRAN binary file blocked with 63 cells to a logical record. If cards are the input medium, it is assumed to be base 32 (0 represented by A, 25 represented by Z and 31 represented by 5) with 6 cells on each card in columns 1 - 72. The key word packets in Examples 1, 2 and 6 would appear on cards as:

Example 1b:

```
AABAAAAAAAAR
```

Example 2b:

```
AADAACAAAAANAAA555AADAAACOY4INWINWIN
```

Example 6b:

```
AAGAAFAAAAAGIKKCPXAAAAAEAA5D2LEEIDM3AAAAAAAAAAADEABOØØWWAIDM3AAAAAAAAAAAK
```

STRUCTURE LEVEL 2.3: SN CONSTRUCTION DETAILS

All SNs are tagged entities, and as mentioned before, the SNs form the skeleton of the data base giving the data base a hierarchy and acting as a template for associated DBEs. Figure 14 presents a model of SN 'SYS.SIMS.XTORS' of Figure 6. In Figure 14 the symbol '@' is interpreted as 'a pointer to'.

Frequently a '@' is followed by a qualifier such as 'B;, 'F', 'L', 'P' or 'S' which have the same connotation as in Structure Level 1.3. The qualifiers have a period separating them from the pointer name. Thus in cell 1 of Figure 14, the pointer @B.STRUCT is a pointer to the parent of the current SN (i.e. SN 'SYS.SIMS'). The first cell (cell no. 0) of a SN is typical of nodes which are retained by cycle numbers. The first cell contains the node type (3 for a current SN, 35 for an old cycle of a SN), the number of cells required to store this node (10 in this example), the cycle this node was created (in this example, CYC = 0), and a pointer to the next cycle or to the original cycle of this node (in this case @CYC LINK points to itself since this is the only cycle of 'XTORS'). The second SN cell contains a backwards pointer (to SN 'SYS.SIMS') and a string of item present bits (IPB). In our example, the number of bits is 2 and the IPB field is 7777_8 since all of the possible cells are present (Structure Level 3 will discuss generation and control of abbreviated nodes).

Cell No.	Cell Type
0	3
1	7
2	9
3	6
4	1
5	1
6	1
7	1
8	1
9	6
10	6

3	#CELLS	CYC	@CYC LINK		
@B.STRUCT			#BITS		IPB
#ATTRIBUTES	#LLH		#DBE		#INWARD@
XTORS					LEVEL#
@F.DBE			@L.DBE		
@F.SUB			@L.SUB		
@F.LLH			@L.LLH		
@P.STRUCT 0			@S.STRUCT 0		
@P.STRUCT 1			@S.STRUCT 1		
DOMAIN					4
DOMCODE					0

Figure 14. SN 'SYS.SIMS.XTORS'

In general the IPB field is interpreted from left to right with a one bit implying a cell is present and a zero bit implying a cell is absent. The third cell contains the number of attributes that the associated DBE contains, the number of linked lists attached to this run, the number of DBEs associated

with this SN, and the number of pointers which point to this SN (in this case this cell contains the values 2, 0, 3, 5). The first four cells form the tag for SN 'SYS.SIMS.XTORS'.

The next three cells (4, 5 and 6) contain pointers to the first and last DBEs associated with SN 'SYS.SIMS.XTORS', the first and last SN in the structure set of SN 'SYS.SIMS.XTORS', and the first and last LLH attached to SN 'SYS.SIMS.XTORS'. In particular the following is true:

@F.DBE points to the DBE with 'DOMCODE = 1' under 'SIM1' in Figure 6.
@L.DBE points to the DBE with 'DOMCODE = 2' under 'SIM2' in Figure 6.
@F.SUB points to the SN 'SYS.SIMS.XTORS.MODELS'.
@L.SUB points to the SN 'SYS.SIMS.XTORS.MODELS'.
@F.LLH points to an order defining node containing 'FIFO' only.
@L.LLH points to an order defining node containing 'FIFO' only.

It is worth noting that the LLHs are chained together and the first entry always contains the set ordering for DBEs associated with this SN only. Cells numbered 7 and 8 in Figure 14 are the structure set membership pointers. There is one pointer cell (2 pointers) corresponding to each ancestor in the SN tree (in this case SN 'SYS' and SN 'SYS.SIMS'). Thus for SN 'SYS.SIMS XTORS' in Figure 6, the subset pointers have the following values:

@P.STRUCT 0 points to SN 'SYS.SIMS'.
@S.STRUCT 0 points to SN 'SYS' (i.e. end of chain).
@P.STRUCT 1 points to SN 'SYS.SIMS' (i.e. beginning of chain.
@S.STRUCT 1 points to SN 'SYS.SIMS.DIODES'.

The last two cells (9 and 10) are the attribute names and value types. In particular one cell indicates an attribute name 'DOMAIN' which is stored as an alpha string (Type = 4), and the other cell indicates an attribute named 'DOMCODE' which is stored as an integer (Type = 0).

The attribute definition cells are used directly as a template for interpreting attribute values stored in the DBE. Thus the attribute definition cells and the attribute value cells are always the same size and in the same order. Because of this, descriptor cells for attribute Types 2 and 3 (double precision and complex) are followed by a void cell

and descriptor cells for arrays are Type 5 cells containing an attribute type and a pointer to an 'AN' primitive cell. The 'AN' cell will follow all the attribute descriptor cells. As an example, the integer array attribute named 'INT', followed by the complex attribute 'COMP' which is itself followed by the real attribute named 'REAL' would have the following attribute descriptor cells:

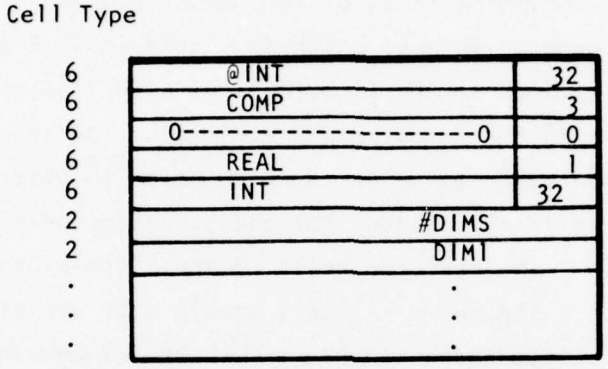


Figure 15 shows the general format of a SN.

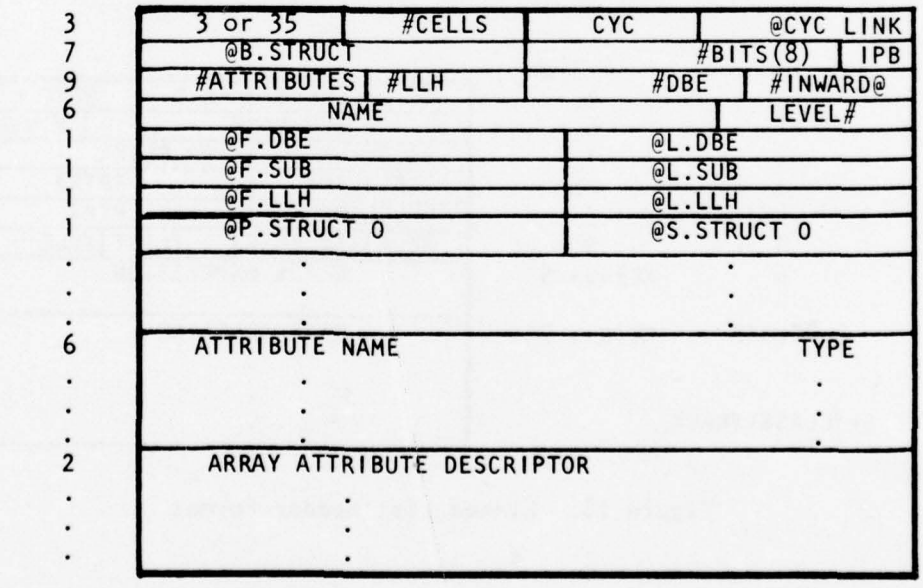


Figure 15. General Structure Node Format

The algorithms required to create and use SNs can be found in Structure Level 3. A node which is attached to a SN (an own node) will be discussed in Structure Level 3.3, since it only deals with construction of SNs.

STRUCTURE LEVEL 2.4: LLH CONSTRUCTION DETAILS

The linked list headers (LLHs) are chained to a SN and they contain either information regarding the ordering of sets, or the membership and ordering of linked lists (LL). The general form of a LLH is displayed in Figure 16. The first cell of a LLH is similar to the first cell of a SN (The node type is 2 for a current cycle and 34 for an out of date cycle). The second cell contains the LL name and the number of pointers which point to this LLH. The third cell of a LLH contains the predecessor and successor pointers for LLHs attached to the same SN. The next two cells (3 and 4) point to the first and last DBE in the LL. The chain pointers within a LL are stored with each linked DBE node. Since there can be many LL chains passing through any DBE the fields FPTR# and LPTR# (relative location in list of LL chain pointers in the first and last DBEs of the LL) are required to identify a particular chain pointer (see Structure Level 2.6). Cell number 5

Cell No.	Cell Type
0	3
1	6
2	1
3	5
4	5
5	9
6	KClass 5
6+#CLASS	KClass 7
5+#CLASS+#RANK	

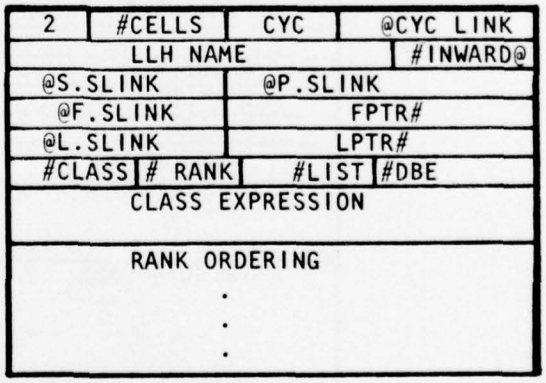


Figure 16. Linked List Header Format

contains the number of cells required to store the CLASS and RANK expressions, the number of ranking phrases in the RANK expression, and the number of DBEs currently in the LL. The LLH ends with two groups of cells. The format of the first group of cells is keyword packet class 5 without the first two cells and contains the class list. The format of the second group of cells is keyword packet class 7 without the first two cells and contains the ranking phrases. (Note the mode of the list is always OAN). Appendix D contains an example of a LLH and Structure Level 3 contains the algorithms for creating and maintaining linked lists.

STRUCTURE LEVEL 2.5: DBE CONSTRUCTION DETAILS

The DBEs are the working part of the data base since all user information is stored in them.

The DBEs contain very little structure information in order to keep them as small as possible. As such the DBEs have a complex tag, and the associated SN must be referenced in order to interpret most of the information contained in a given DBE. The general form of a DBE is presented in Figure 17. The first two cells of a DBE are similar to the first two cells of a SN. Here the node type is 5 (37 for an outdated DBE). The pointer '@B.STRUCT' points to the SN associated with this DBE. Cell number 2 contains the sibling ring pointers. In particular '@P.DBE' points

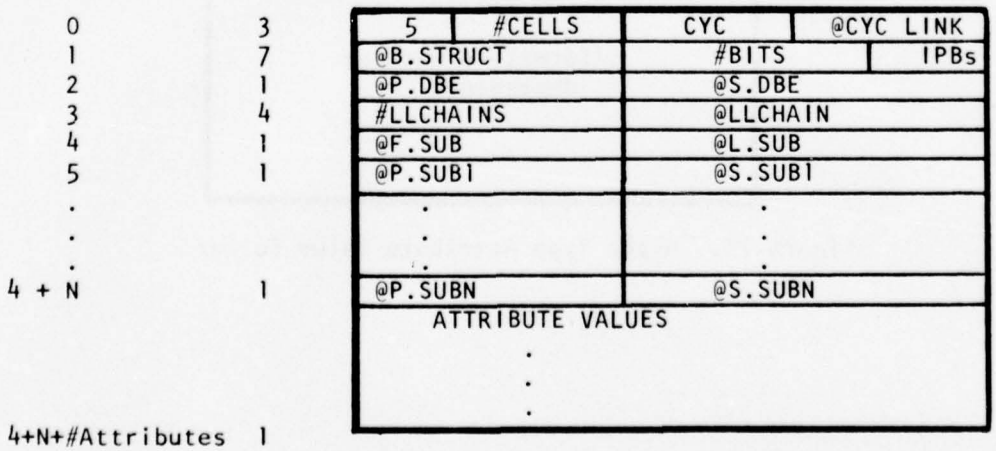


Figure 17. Data Base Entry Node Format

to the particular DBE associated to the same SN which was created just prior to this DBE. Similarly '@S.DBE' points to the DBE created just after this DBE which is associated with the same SN. Cell number 4 contains pointers to the first and last DBE in the ordered set belonging to this DBE. The fourth cell of a DBE contains the number of LL chain slots and a pointer ('@LLCHAIN') to a node containing the LL chain pointers. The next N cells contain pointers for each of the possible sets this DBE may belong to (N is the level of this DBE). This association of nodes to sets has been discussed in Structure Level 2.3.

Single attribute values of type INTEGER or REAL are stored in Type 2 cells. Single DOUBLE PRECISION and COMPLEX values each require two Type 2 cells. Alpha values are stored in the internal character code of the machine on which the data base resides. If the string requires less than 52 bits, (6 or 8 characters) the string is stored along with the number of characters in a Type 6 cell. Otherwise the number of characters and a pointer to the character string are stored in a Type 5 cell. The alpha string is placed in an alpha node as displayed in Figure 18. The first two cells are of a familiar format. Note the node types for current and outdated alpha nodes are 9 and 41 respectively.

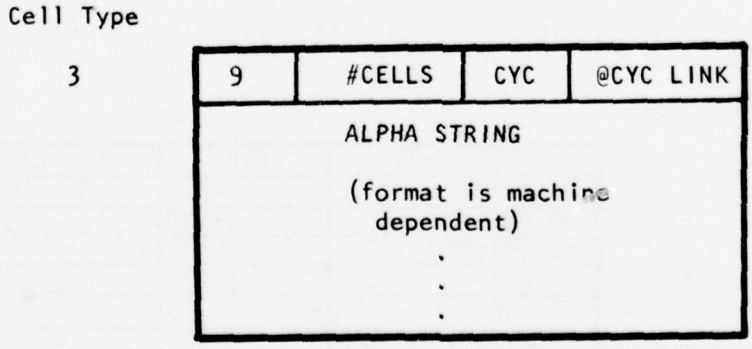


Figure 18. Alpha Type Attribute Value Format

A bit string is stored in the same fashion as an alpha string except it is not machine dependent. Any of the attribute value types mentioned so far may be stored as arrays. For an array the following Type 1 cell, called an array entry, is stored in the DBE node:

@ARRAY DESCRIPTION	@ARRAY VALUES

Here the 'ARRAY DESCRIPTION' and 'ARRAY VALUES' are nodes of the form shown in Figures 19a and 19b respectively. The first cell of both nodes is the standard cycle description. Note an array description node is Type 7 (and 39). In Figure 19a the number of indices and the length of each dimension are expressed as integers.

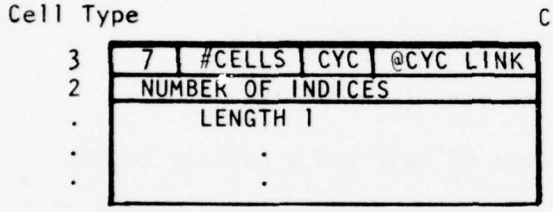


Figure 19a. Array Description

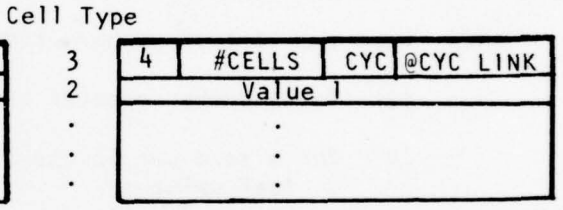


Figure 19b. Array Values

The values for data base name attributes are stored as a singly indexed array of length 5. The five alpha strings represent the file name (i.e. DBN), read key, write key, alter key, and system key. As an example, if the data base name is 'TEST', the write key is 'DONT', and the remaining keys are unspecified. The attribute value would be represented by Figure 20.

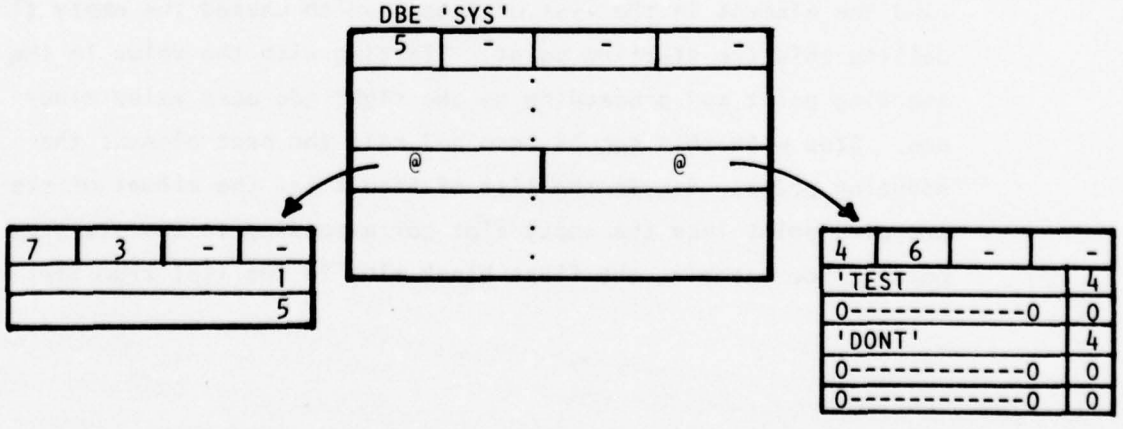


Figure 20. Data Base Name Attribute Value Format

The most complex attribute type is a ragged table. The method of constructing a ragged table will not be discussed until Structure Level 3; however, the general form of a ragged table will be discussed presently. Similar to all attribute types, a single Type 1 cell is included in the DBE node. This cell points to a Ragged Table Node (RTN). The RTN is a contiguous group of Type 9 cells. Considering the ragged table of Figure 9, the RTN is intuitively constructed as follows:

(1) Perform a pre-order traversal of the ragged table (tree) writing down the number of outgoing branches. For our example this results in the following sequence

3 2 0 2 0 0 1 0 2 1 1 2 0 2 0 0 0

(2) Scan the list of numbers from left to right and

(a) for a number greater than zero leave a slot

(b) for a zero put in the type of leaf and a pointer to the leaf value

Step 2 results in the following lists:

3 2 0 2 0 0 1 0 2 1 1 2 0 2 0 0 0 (copies from step 1)

_ _#A _#B#C _#D _ _ #E _ #F#G#H

where #A represents both the attribute type and the pointer to the attribute value of A.

(3) Scan the list of Step 2 from left to right and for each empty slot find the element in the list of Step 1 which caused the empty slot, calling this the starting point. Starting with the value in the starting point and proceeding to the right add each value minus one. Stop when this sum is zero and call the next element the stopping point. Now in the list of Step 2 set the offset of the stopping point into the empty slot corresponding to the starting point. For example, the first blank slot in the list from Step 2

corresponds to the 3 in the list of Step 1. Summing each of the elements to the right we obtain the following sequence:

3,4,3,4,3,2,2,1,2,2,2,3,2,3,2,1,0.

Thus the stopping point is just past the end of the list (#H), and the list of Step 2 becomes:

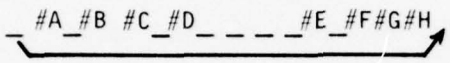


Figure 21 shows the resultant list.

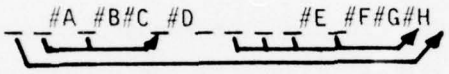


Figure 21. Example of Ragged Table Construction

When this list is stored into a table each slot corresponds to one subcell (15 bits) of a Type 9 cell, and each leaf consists of one subcell (15 bits) for the attribute type followed by two consecutive subcells (30 bits total) for a pointer to the attribute value. The resultant table is shown in Figure 22. The first two cells of a RTN as shown in Figure 22 form the header. The first cell contains type, size and cycle information which has been discussed previously. Here the 8 implies a current RTN and a 40 would imply an out-of-date cycle. The second cell will be discussed in detail in Structure Level 3, but for now it suffices to mention the second cell is used to control overhead during construction and alteration of a ragged table.

Subcell No.					Cell Type
	8	#CELLS	CYC	@CYC LINK	3
	#Subcells	#*	# ?	Max Subs	9
0	33	12	#A	@A(1/2)	9
4	@A(2/2)	12	#B	@B(1/2)	9
8	@B(2/2)	#C	@C		9
12	16	#D	@D		9
16	33	30	30	30	9
20	#E	@E		30	9
24	#F	@F		#G	9
28	@G		#H	@H(1/2)	9
32	@H(1/2)				

Figure 22. Ragged Table Storage Format

A tabular representation of Figure 21 is shown starting with the third cell. The cells containing numbers correspond to the pointers of Figure 21. #A is the complement of the attribute type of leaf A, and @A (requiring two subcells) is a pointer to the value cells of leaf 'A'. During construction of a ragged table, subtrees are not always nested in the RTN; this occurrence is flagged by an attribute type of 256 (complemented). Also if a ragged table exceeds its allocated space, DBMS attaches new ragged tables to the original tree with an attribute type of 512.

Before continuing, it is necessary to examine the properties of the RTN of Figure 22. Notice that each branch of the ragged table appears sequentially. For example, the leftmost branch of the ragged table (with leaves A, B and C) is contained in subcells 1 through 11 and the branch with leaves B and C is contained in subcells 5 through 11. Also each branch (excluding leaves) is preceded by a pointer to the first cell of the next branch. In order to demonstrate this better, let us follow the branches to leaf C. Starting at node 0 we find a pointer to subcell 33 (or 'a pointer to 33' for short). If we follow this pointer we will skip over the entire ragged table (considered as one large branch). Ignoring the first pointer we go to subcell 1 and find a pointer to 12. Following this pointer would skip over the leftmost branch, but since C is on the leftmost branch we also ignore this pointer and move on. Subcell 2 is flagged as being a leaf, but we need to take the second branch so we ignore the leaf and come to subcell 5. Here again is a pointer to skip over the branch we want, so we ignore it and move on to subcell 6. This is also a leaf we don't want (B), so ignoring it we come to subcell 9 which is leaf C. The following points are now apparent:

(1) Motion in the table is always down (increasing subcell numbers).

(2) Motion in the tree is accomplished by finding the proper branch and then moving down a level. Then repeat until the desired node is reached.

(3) The pointers which are not taken provide a decreasing maximum subcell number. For example, when 'C' was found this maximum

was 12; if we had tried to skip 'C', we would have been at subcell 12 which is an error. Thus requesting a non existent branch is detectable.

(4) Since leaves are distinct, an attempt to consider a leaf as a branch node is detectable. This would correspond to moving down a non-existent branch.

Branch and leaf nodes are referenced in the same format as simple arrays. This notation can be converted to an intuitive notation consisting of combinations of the operators +1 (move down a level) and * (skip a branch) by the following algorithm:

Move through indices from left to right and for each index

(a) Place a +1 at end of current expression and enclose the result in parentheses.

(b) Place (index-1) *'s in front of the current expression.

As an example the leaves of Figure 6 would be:

(1) leaf A with indices (1,1) becomes ((+1)+1)

(2) leaf B with indices (1,2,1) becomes (*((+1)+1)+1)

(3) leaf C with indices (1,2,2) becomes *(*((+1)+1)+1)

(4) leaf D with indices (2,1) becomes (*(+1)+1)

(5) leaf E with indices (3,1,1,1,1) becomes ((((**(+1)+1)+1)+1)+1)

(6) leaf F with indices (3,1,1,1,2,1) becomes (*((((**(+1)+1)+1)+1)+1)+1)

(7) leaf G with indices (3,1,1,1,2,2) becomes *(*((((**(+1)+1)+1)+1)+1)+1)

(8) leaf H with indices (3,2) becomes *(**(+1)+1)

The motion produced by +1 and * would be:

+1 current subcell + 1 if the current subcell is a pointer (not a leaf)
error if current subcell is a leaf

* indirect address if current subcell is a pointer (not a leaf)
current subcell + 3 if current subcell is a leaf

Note that DBMS performs the above ragged table algorithms in a much condensed form.

STRUCTURE LEVEL 2.6: LINKED LIST CHAIN (LLC) NODES

Figure 17 contains the pointer '@LLCHAIN' which points to a LLC node. The general format of a LLC node is displayed in Figure 23. The first two cells are standard and will not be discussed. The chain links require a pointer to LLC node and a pointer number (or offset). Thus if we have a pointer to the LLC node of Figure 23 and an offset of 3 the next element in the chain will be the LLC node pointed by '@LLC3' and the new offset will be '#LLC3'. Each offset and pointer is considered a single item in a LLC node.

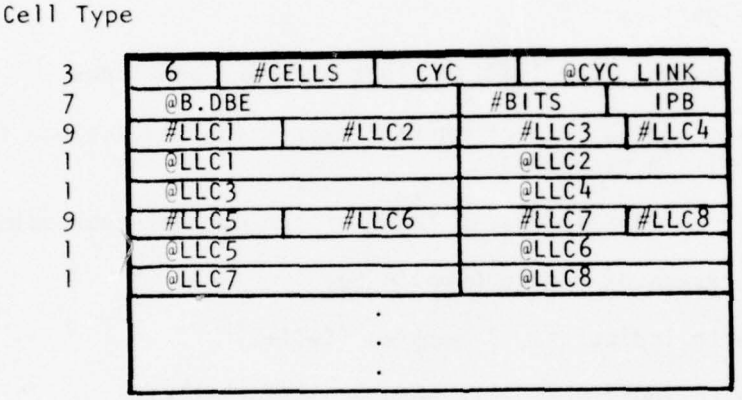


Figure 23. Linked List Chain Format

STRUCTURE LEVEL 2.7: THE DATA BASE NODE

The first three cells of a data base are always of the format represented in Figure 24. The first cell identified the file as a data base generated on a CDC 6600/7600. The left half of the second cell contains a pointer to SN 'SYS'. The remaining fields are discussed in Structure Level 3.

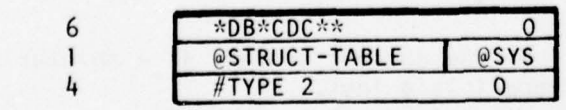


Figure 24. Data Base Node Format

Note that the fields @STRUCT-TABLE and #TYPE 2 cells are included to insure that data bases are upward compatible with future versions of DBMS. Presently these two fields are zeros.

SECTION IV

DBMS ALGORITHMS

STRUCTURE LEVEL 3.0: DBMS MAINTENANCE

Structure Level 3 contains the DBMS algorithms and functional descriptions of the main procedures used by DBMS. As such structure level 3 can be used by a knowledgeable systems programmer to perform maintenance tasks on DBMS and associated data bases. Although DBMS maintains a high level of system sanity within a data base, a data base can nevertheless be partially destroyed by a hardware failure during a disk write, to name only one example. Conceivably the systems programmer can detect and correct the incorrect data and thus save a large data base. Similarly, in a data base which has a large number of 'bad areas', a systems programmer can transfer valid information to a new data base. Also, as will be seen, the knowledgeable systems programmer can determine which pointers are saved in a data base (i.e. '@F.SUB' might be stored in a DBE and '@L.SUB' is not). Data bases remain compatible with DBMS regardless of which pointers are saved in the data bases and which pointers are used by DBMS.

STRUCTURE LEVEL 3.1: LONG INPUT DECOMPOSITION

When the input mode is LONG, the card images are translated to the internal command form by a table driven LR(k) parser which calls a table driven lexical analyzer. The BNF grammar for the DBMS is presented in Table 7. The format of the translator output is presented in Structure Level 2.2.

STRUCTURE LEVEL 3.2: UTILITY FUNCTIONS

The algorithms for the Type 1 (utility) functions are presented in Tables 8 through 15. Subtleties of the Type 1 functions are discussed below.

Pointers in the data base may point to an out of date cycle of a node, in which case the cycle link must then be followed until the current node

is found. Figure 25a shows several cycles of a node and an out of date pointer. Figure 25b shows the same pointer after updating. Before an out of date node can be purged, the cycle links must be adjusted to bypass the node and all inward pointers must be set to point at the current cycle of the node. This restriction only affects the function COPY (Table 10) of all the Type 1 functions. Because the data base is paged, the page containing the tag of a node must be locked into core before any part of the node can be referenced into core. Then if the node is out of date, the cycle link must be followed. This is accomplished by determining the location of the next cycle by unlocking the page of the out of date node and locking the new page into core. This process is repeated until the current node is located. It might then be necessary to lock an additional page to access the desired part of a node.

STRUCTURE LEVEL 3.3: STRUCTURE FUNCTIONS

The algorithms for the three Type 2 functions are presented in Tables 16 through 18. These functions deal with the generation and control of the structure tree and the generation and removal of linked list headers and initial chains.

A structure node may be redefined by the structure function. This results in a new cycle of the SN. If all the attributes of the out of date SN are present and in the same relative location within the new attribute list, the associated DBEs are retained as is. Otherwise, the associated DBE are flagged for deletion. The resultant effect of flagging DBEs for deletion is discussed further in Structure Level 3.4.

For each structure command which contains an own keyword, there is created an own node. This node is used during the construction of a structure tree and as such was not mentioned in Structure Level 2. Figure 26 shows the general format of an own node (ON). The node type is 1 for an ON.

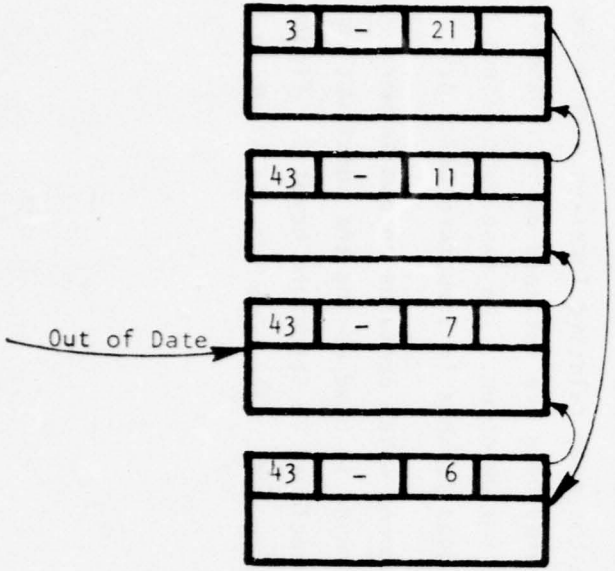


Figure 25a. Pointer to an Out of Date Cycle

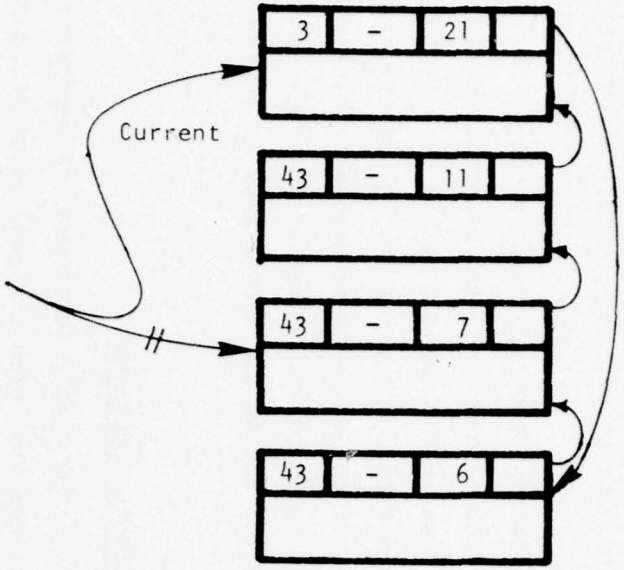


Figure 25b. Pointer to the Current Cycle

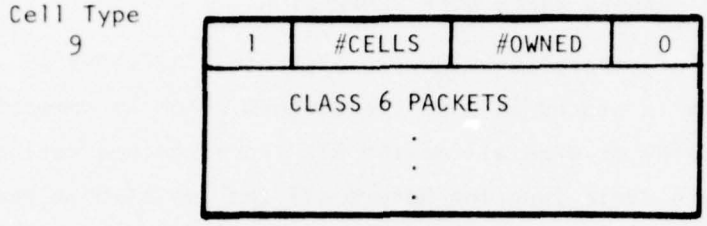


Figure 26. Own Node Format

A pointer to an ON is placed in a SN as @P.STRUCTN and @S.STRUCTN where N is the level of the SN (See Figure 15).

As was mentioned in Structure Level 2.3, not all pointers must be included in a given node. However, all of the chains must be complete. Thus if a SN has one or more DBE which do not have a @P.DBE, then all the DBE associated with that SN must have @S.DBE present. Further, if in specifying a node, a motion phrase requiring an absent chain pointer is given, then DBMS will proceed in the reverse direction till the proper node is found. Which pointers are to be placed in a node is specified in the BLOCK DATA by setting logical variables. These variables are located under the comment stating 'POINTER CONTROL'. When choosing these pointers, it should be remembered DBMS primarily uses forward (or downward) pointers with the exception that $B is used frequently.

STRUCTURE LEVEL 3.4: DATA CONTROL FUNCTIONS

The algorithms for the eight Type 3 functions are presented in Tables 19 through 26. These functions deal with the storage and retrieval of data within the data base plus the control of DBE sets. The creation of a new data node is simply tying an empty node into the chain of DBEs associated with the specified SN. This new node has all of the attribute value IPBs turned off. These IPBs will be turned on as needed by the store function. A similar action is performed for the set and link pointers in that they are initially null and set as needed. Structure Level 3.5 will discuss the generation of ragged table nodes. The other type of attribute values are created in a straight forward manner.

STRUCTURE LEVEL 3.5: RAGGED TABLE NODE GENERATION

The algorithms for generating RTNs are presented in Tables 27 and 28. During creation a RTN is stored in a scattered form which is compacted as the RTN becomes unwieldy or when all of the RTN leaves become defined. If a RTN is archived by a cycle function before all the leaves have been defined then that cycle is stored in the less efficient scattered form. When the first node is defined, presumably a branch, a complete tree is defined with a leaf for each subtree defined. The attribute type of these temporary leaves is an undefined continuation. As subsequent branches are defined they are stored as complete and disjoint subtrees and the corresponding undefined continuation is converted to a defined continuation pointing to the new subtree. Note however, if the leaf being continued is the last leaf of the last subtree in the RTN, then the new definition is appended directly to this last subtree. The number of defined continuations is monitored there by giving a measure of the excess overhead in the RTN. When the RTN overhead becomes too large, the RTN is compacted. When a leaf of the RTN is defined, the corresponding undefined continuation is replaced by the real leaf. Figure 27 shows the RTN as defined in Structure Level 1.4 just prior to compacting. Note that a leaf of type defined continuation is indicated by three cells, the first is an * and the last two

Subcell				
0	10	8	0	10
4	*	0	21	*
8	0	25	21	#A
12	@	A	21	#B
16	@	B	#C	@
20	C	25	#D	@
24	D	32	*	0
28	32	#H	@	H
32	45	45	45	#E
36	@	E	45	#F
40	@	F	#g	@
44	G	-	-	-

Figure 27. Scattered Ragged Table Node

contain a pointer to the subtree. The compaction algorithm performs a preorder transversal of the scattered tree and generates a compacted tree as shown in Figure 22. As indicated in Tables 27 and 28, the process is complicated by the fact that a RTN is allocated storage locations in increments. These increments are treated as permanent subtrees and the compaction algorithm only treats one of these trees at a time. The complications caused by the incremental storage allocation is one of bookkeeping and is not discussed further here. The constants controlling the generation of RTNs are located in the BLOCK DATA under the title 'RTN DEFINITIONS'. These constants define the size of the incremental storage used and the maximum overhead allowed.

```
cprog := coms EOF
coms := command
coms := coms command
command := function EOC
function := COMPRESS
function := COMPRESS dbnfields
function := COPY copfields
function := CYCLE
function := CYCLE dbnfields
function := DISPLAY
function := DISPLAY disfields
function := INITIAL
function := INITIAL dbnfields
function := LIST
function := LIST lisfields
function := RESTORE
function := RESTORE resfields
function := RETRIEVE
function := RETRIEVE dbnfields
function := LINK linfields
function := STRUCTURE strfields
function := UNLINK , llkey
function := CREATE crefields
function := ENTER entfields
function := FIND , dbekey
function := STORE
function := STORE stofields
function := rdr
function := rdr rdrfields
rdr := RETURN
rdr := DELETE
rdr := REMOVE
function := UNFILE
function := UNFILE unffields
function := END DATA
function := MODE
function := MODE modfields

copfields := copk
copfields := copfields cork
copk := , TO = filename
copk := , LINKS = SAVE
copk := , LINKS = DELETE
copk := , INCLUDE = snlist
copk := , EXCLUDE = snlist
copk := copkf ( cycleno )
copk := copkf
copkf := , FROM = filename
copk := , KEY = rwakey
copk := , SYSKEY = syskeykey
```

Table 7. BNF for DBMS

```
disfields := disk
disfields := disfields disk
disk := dbnfields
disk := disstart
disk := outloc
disk := , LISTS
disk := , SETS
disstart := , snkey
disstart := , BELOW = snnam
disstart := , ABOVE = snnam
disstart := , ALL = snnam
disstart := , dbekey
lisfields := lisk
lisfields := lisk lisk
lisk := dbnfields
lisk := outloc
resfields := resk
resfields := resk resk
resk := dbnfields
resk := , TAPE = filename

linfields := linke
linfields := linfields linke
linke := , llkey
linke := , CLASS = ( exp )
linke := , rankey
strfields := strk
strfields := strfields strk
strk := , snkey
strk := , ATTRIBUTE = alistl
alistl := ( alists)
alists := alists , alistelm
alists := alistelm
alistelm := aqual an
alistelm := aspecqual aspecn
alistelm := aqual ( alist2)
alistelm := aspecqual ( aspeclist )
aqual := INTERER
aqual := REAL
aqual := COMPLEX
aqual := DOUBLE
aqual := ALPHA
aqual := bitqual
aqual := bitqual ( width )
bitqual := BITS
aspecqual := DB
aspecqual := RAGGED
width := integer
strk := , MEMBER = snlist
strk := , OWN = snlist
```

Table 7. BNF for DBMS (Continued)

```
strk := , rankey
strk := , DELETE

crefields := crek
crefields := crek crek
crek := , snkey
crek := , valuekey
entfields := entk
entfields := entfields entk
entk := , dbekey
entk := , orderkey
entk := , setkey
rdrfields := rdrk
rdrfields := rdrfields rdrk
rdrk := , dbekey
rdrk := , NAME = alist2
rdrk := outloc
stofields := stok
stofields := stok stok
stok := , dbekey
stok := , valuekey
unffields := unfk
unffields := unfk unfk
unfk := , orderkey
unfk := , setkey

modfields := modk
modk := , modein
modfields := modk modk
modein := LONG
modein := SHORT
modk := modloc
modloc := , CORE = location ( size )
modloc := , FILE = filename
modloc := , CARD

dbekey := DBE = dbenam
dbnfields := dbnk
dbnk := , DBN = filename
dbnk := , KEY = rwakey
dbnk := , SYSKEY = syskeykey
llkey := LL = llnam
llnam := llprim $H
llnam := llprim $H nodename
llprim := nodename
llprim := nodename samesides
llprim := samesides
llprim := llprim llprimcomplex
llprimcomplex := lldbeside $B
lldbeside := $H
lldbeside := $H nodename
```

Table 7. BNF for DBMS (Continued)

```
lldbeside := *
orderkey := ORDER = FIRST
orderkey := ORDER = LAST
orderkey := ORDER = BEFORE dbenam
orderkey := ORDER = AFTER dbenam
outloc := , CORE = location ( size )
outloc := , FILE = filename
outloc := , PRINT
rankey := RANK = ( ranks )
ranks := ranks ranker
ranks := ranker
ranker := HIGH an
ranker := LOW an
ranker := FIFO
ranker := LIFO
setkey := SET = dbenam
snkey := SN = snnam
valuekey := VALUE = ( vpairs )
vpairs := an = const
vpairs := vpairs , an = const

location := integer
size := integer
snlist := ( sns )
sns := snnam
sns := sns , snnam
cycleno := integer
const := integer
const := real
const := complex
const := double
const := dbname
const := alpha
dbname := filename
dbname := filename ( dbargl )
dbname := filename ( dbargl , dbarg2 )
dbargl := SYSKEY = syskeykey
dbarg2 := KEY = rwarey
rwakey := rkey wkey akey
rwakey := rkey wkey
rkey := nodename /
rkey := /
wkey := /
wkey := nodename /
akey := nodename
syskeykey := alpha
alist2 := an
alist2 := alist2 , an
aspeclist := aspecn
aspeclist := aspeclist , aspecn
```

Table 7. BNF for DBMS (Continued)

```
snnam := constructname
snnam := constructname samesides
snnam := samesides
snnam := sncomplex
snnam := snnam sncomplex
sncomplex := dbeside $B
sncomplex := dbeside samesides $B
dbenam := constructname
dbenam := samesides
dbenam := constructname samesides
dbenam := constructname samesides dbeside
dbenam := dbecomplex
dbenam := dbenam dbecomplex
dbecomplex := $B dbeside
dbecomplex := $B samesides dbeside
dbeside := listent
dbeside := listent constructname
listent := $H
dbeside := *
constructname := nodename
samesides := sameside
samesides:= samesides sameside
sameside := $F
sameside := $L
sameside := $S
sameside := $P
sameside := preexp exp
prexp := ,
prexp := /

exp := nodename
exp := ( bolean )
bolean := bterm
bolean := bolean .OR. bterm
bterm := bfactor
bterm := bterm .AND. bfactor
bfactor := rel
bfactor := ( bolean )
bfactor := .NOT. bfactor
rel := aexp relop aexp
aexp := alph
aexp := integer
aexp := real
aexp := complex
aexp := double
aexp := filename
aexp := an
an := nodename
an := namenode ( dims )
namenode := nodename
dims := integer
```

Table 7. BNF for DBM$ (Continued)

```
dims := dims , integer
aspecn := nodename
relop := .LT.
relop := .LE.
relop := .EQ.
relop := .NE.
relop := .GT.
relop := .GE.
```

Notes:

1. The field 'filename' is defined in the general discussion of Type 1 functions in Appendix A.

2. The field 'nodename' is defined in Structure Level 1.2.

3. The field 'integer' is defined in Appendix A in the discussion of the Type 3 function CREATE.

4. The field 'real' is any ANSI FORTRAN real constant.

5. The field 'complex' is any ANSI FORTRAN complex constant.

6. The field 'double' is any ANSI FORTRAN double precision real constant.

7. The field 'alpha' is defined in Appendix A in the discussion of the Type 3 function CREATE.

8. The fields 'EOF' and 'EOC' are automatically supplied by DBMS, and must not be coded by a user.

Table 7. BNF for DBMS (Concluded)

```
SUBROUTINE INITIAL
*IF THERE IS A CURRENT DATA BASE THEN CLEAN UP THE CURRENT DATA BASE
*IF DATA BASE NAME IS PRESENT
*THEN
   *ATTACH DATA BASE FILE TO CURRENT RUN
   *IF FILE IS ALREADY A DATA BASE AND ALTER KEY DOES NOT MATCH THEN ERROR
*ELSE
   *ATTACH A TEMPORARY DATA BASE CALLED DBN
*GENERATE DATA BASE HEADER NODE AND SY'SYS'
*RETURN
```

Table 8. DBMS Function Initial

```
SUBROUTINE RETRIEVE
*IF THERE IS A CURRENT DATA BASE THEN CLEAN UP THE CURRENT DATA BASE
*IF DATA BASE NAME IS NOT PRESENT THEN USE DEFAULT = DBN
*ATTACH DATA BASE FILE TO CURRENT RUN
*IF THE FILE IS NOT A DATA BASE THEN ERROR
*COMPARE AND SET PERMISSION PARAMETERS
*RETURN
```

Table 9. DBMS Function Retrieve

```
SUBROUTINE COPY
   .
*IF THE CURRENT DATA BASE NAME IS DIFFERENT FROM THE NEW NAME (KEYWORD 'FROM')
*THEN
   *PERFORM RETRIEVE FUNCTION ON NEW DATA BASE
*IF TO FIELD IS A DATA BASE AND THE WRITE OR ALTER KEY DOES NOT MATCH THEN ERROR
*IF NO CYCLE NUMBER SPECIFIED
*THEN
   *SET CYCLE = CURRENT CYCLE
*ELSE
   *VERIFY CYCLE GIVEN IS IN DATA BASE
*IF 'FROM' DATA BASE NAME IS NOT THE SAME AS THE 'TO' DATA BASE NAME
*THEN
   *IF 'TO' FIELD IS TAPE THEN SUBSTITUTE TEMPORARY FILE FOR 'TO' DATA BASE NAME
   *IF BOTH INCLUDE AND EXCLUDE FIELDS ARE PRESENT THEN ERROR
   *PERFORM PRE ORDER SCAN OF STRUCTURE TREE AND FOR EACH SN
      *IF((SN IS IN INCLUDE LIST OR AN ANCESTOR OF AN INCLUDED SN) AND (SN IS NOT
       AN EXCLUDED NODE OR A DESCENDENT OF AN EXCLUDED NODE) OR (NO INCLUDE/EXCLUDE
       LIST))
      *THEN
         *COPY SPECIFIED CYCLE(S) OF SN AND LLH IF PRESENT USING STRUCTURE LINK
          FUNCTIONS
         *SCAN DBES ASSOCIATED WITH SN
            *COPY SPECIFIED CYCLE(S) OF DBES IF PRESENT USING CREATE AND ENTER
             FUNCTIONS
         *FLAG ENTRY IN INCLUDE/EXCLUDE LIST IF FOUND
   *PRINT NAME OF ALL SNS NOT FOUND IN INCLUDE/EXCLUDE LIST
   *IF 'TO' FIELD IS A TAPE
   *THEN
      *PERFORM COPY FUNCTION FROM TEMPORARY FILE TO TAPE
*ELSE
   *SCAN STRUCTURE TREE
   *IF NOT EXCLUDED BY INCLUDE/EXCLUDE LIST
   *THEN
      *SCAN ASSOCIATED DBE
         *UPDATE ALL POINTERS TO PROPER CYCLE
         *DESIGNATE ALL UNWANTED CYCLES OF NODES AS AVAILABLE SPACE
   *ELSE
      *SCAN STRUCTURE SUBTREE WITH ROOT SPECIFIED BY KEYWORD 'SN'
         *FLAG NODE AS DELETED
         *SCAN ASSOCIATED DBE
            *PERFORM DELETE FUNCTION
*PERFORM COMPRESS OF DATA BASE
*RETURN
```

Table 10. DBMS Function Copy

```
SUBROUTINE CYCLE
*IF CURRENT DATA BASE NAME IS DIFFERENT FROM NEW NAME
*THEN
   *PERFORM RETRIEVE FUNCTION ON NEW DATA BASE
*IF ALTER KEY NOT PROPER THEN ERROR
*FIND CYCLE ATTRIBUTE ARRAY ON DBE 'SYS'
*IF CYCLE TABLE IS FULL
*THEN
   *SCAN DATA BASE
      *UPDATE ALL POINTERS
      *IF CYCLE IS LESS THAN NUMBER CYCLES TO LOSE
      *THEN
         *FLAG AS AVAILABLE
      *ELSE
         *REDUCE CYCLE NUMBER BY NUMBER CYCLES TO LOSE
   *PRINT CYCLES TO BE PURGED
   *ADJUST CYCLE TABLE TO SHOW CURRENTLY RETAINED CYCLES
*ENTER NEW CYCLE DATE AND TIME
*RETURN
```

Table 11. DBMS Function Cycle

```
SUBROUTINE RESTORE
*IF CURRENT DATA BASE NAME IS DIFFERENT FROM NEW NAME
*THEN
   *PERFORM RETRIEVE FUNCTION ON NEW DATA BASE
*IF ALTER KEY DOES NOT MATCH NEW THEN ERROR
*READ SHORT HEADER (DATA BASE NODE)
*IF SAME MACHINE (I.E. IBM or CDC)
*THEN
   *COPY TAPE TO DATA BASE
*ELSE
   *COPY TAPE TO TEMPORARY RANDOM ACCESS
   *SCAN RANDOM ACCESS AND
      *RECREATE DATA BASE USING STRUCTURE, LINK, CREATE AND ENTER FUNCTIONS
*RETURN
```

Table 12. DBMS Function Restore

```
SUBROUTINE LIST
*IF CURRENT DATA BASE NAME IS DIFFERENT FROM NEW NAME
*THEN
   *PERFORM RETRIEVE FUNCTION ON NEW DATA BASE
*IF READ KEY DOES NOT MATCH THEN ERROR
*FIND CYCLE ATTRIBUTE ARRAY FOR DBE 'SYS'
*PRINT TIME AND DAY OF LAST (SYSTEM SPECIFIED) CYCLES
*RETURN
```

Table 13. DBMS Function List

```
SUBROUTINE DISPLAY
*IF CURRENT DATA BASE NAME IS DIFFERENT FROM NEW NAME
*THEN
   *PERFORM RETRIEVE FUNCTION ON NEW DATA BASE
*IF READ KEY DOES NOT MATCH THEN ERROR
*IF TWO OR MORE KEYWORDS FROM THE GROUP (SN, BELOW, ABOVE, ALL OR DBE) ARE
 PRESENT THEN ERROR
*IF PRINT IS SPECIFIED
*THEN
   *DO CASE FOR KEYWORD CHOSEN
   *CASE1 SN
      *FIND SN
      *FOLLOW CYCLE POINTER TO OLDEST CYCLE
      *SCAN ALL CYCLES
         *PRINT THE ATTRIBUTE NAMES AND VALUE TYPE
      *PRINT THE NAMES OF THE SN'S CHILDREN
      *PRINT THE NUMBER OF LINKED LISTS
      *IF LISTS OPTION PRESENT
      *THEN
         *PRINT CLASS FIELD
         *PRINT RANK FIELD
   *CASE2 BELOW
      *SCAN STRUCTURE TREE WITH ROOT SPECIFIED
         *PRINT NODE NAME AND LEVEL
         *PRINT CURRENT ATTRIBUTE NAME AND VALUE TYPES
         *THE NUMBER OF LINKED LISTS
   *CASE3 ABOVE
      *FIND SN
      *SCAN BACKWARDS POINTERS UP TO BUT NOT INCLUDING NODE SYS
         *PRINT CURRENT ATTRIBUTE NAMES AND VALUE TYPE
         *PRINT NODE NAME
         *PRINT THE NUMBER OF LINKED LISTS AND DBE'S
   *CASE4 ALL
      *FIND SN
      *PRINT FIRST SIX ATTRIBUTE NAMES AND VALUE TYPES
      *SCAN ASSOCIATED DBE
         *PRINT FIRST SIX ATTRIBUTE VALUES
   *CASE5 DBE
      *ASSOCIATED SN'S GENERIC NAME
      *FOLLOW CYCLE POINTER TO OLDEST CYCLE
      *SCAN CYCLES
         *PRINT CYCLE INFORMATION
         *ATTRIBUTE NAME AND VALUE
         *IF VALUE PRINT IS SMALL THEN PRINT ATTRIBUTE VALUE
      *IF ALL OPTION IS PRESENT
      *THEN
         *SCAN DBE SET SPECIFIED
            *PRINT FIRST SIX ATTRIBUTE NAMES
            *IF VALUE PRINT IS SMALL THEN PRINT VALUE
*ELSE
   *TRANSFER NODES TO SPECIFIED MEDIUM
   *IF CORE OVERFLOW THEN ERROR
```

Table 14. DBMS Function Display

```
SUBROUTINE COMPRESS
*ATTACH TEMPORARY DATA BASE
*PERFORM COPY FROM SPECIFIED DATA BASE TO THE TEMPORARY DATA BASE
*BINARY COPY FILE BACK
*RETURN
```

Table 15. DBMS Function Compress

```
SUBROUTINE LINK

*IF THERE IS NOT A CURRENT DATA BASE
*THEN
   *PERFORM RETRIEVE FUNCTION ON DEFAULT DBN
   *IF DEFAULT DBN IS NON RETRIEVABLE
   *THEN
      *PERFORM INITIALIZATION OF DEFAULT DBN
*FIND PARENT SN
*IF A LLH BY SAME NAME ALREADY PRESENT
*THEN
   *CREATE NEW CYCLE OF LLH
   *IF OLD CYCLE NUMBER IS SAME AS CURRENT AND NO INWARD POINTERS
   *THEN
      *REMOVE OLD LLH FROM CHAINS
      *SET STATUS OF NODE TO AVAILABLE
*ELSE
   *CREATE NEW LLH ON THE END OF THE LLH CHAIN
*IF NO LLH NAME GIVEN THEN SET NAME TO DEFAULT
*PLACE NAME IN NEW LLH NODE
*PLACE CLASS EXPRESSION INTO LLH NODE
*SET LL CHAIN POINTERS TO THIS LLH
*PLACE RANK EXPRESSION INTO LLH NODE
*SCAN DESCENDANTS OF THE LLH'S PARENT SN
   *COMPARE ATTRIBUTE NAME PRESENT WITH THOSE REQUIRED FOR CLASS EXP
   *IF REQUIRED NAMES ARE PRESENT
   *THEN
      *SCAN ASSOCIATED DBE
         *IF CLASS EXPRESSION MATCHES
         *THEN
            *FIND START OF CHAIN
            *USE RANK EXPRESSION TO LOCATE POSITION IN LL FOR THIS DBE
            *INSERT DBE INTO CHAIN
*PLACE NUMBER OF DBE INTO LLH HEADER
```

Table 16. DBMS Function Link

```
SUBROUTINE STRUCTURE
*IF THERE IS NOT A CURRENT DBN
*THEN
   *PERFORM RETRIEVE FUNCTION ON DEFAULT DBN
   *IF DEFAULT DBN IS NON RETRIEVABLE
   *THEN
      *PERFORM INITIALIZATION OF DEFAULT DBN
*IF DELETE OPTION PRESENT
*THEN
   *IF ATTRIBUTE, MEMBER, OWN OR RANK FIELD PRESENT THEN ERROR
   *FIND SN
   *SCAN STRUCTURE TREE WITH THIS NODE AS A CHILD
   *FLAG SN FOR DELETION
   *SCAN ASSOCIATED DBE
      *PERFORM DELETE FUNCTION
   *SCAN LINKED LISTS HEADERS
      *PERFORM UNLINK FUNCTION FOR THIS LLH
   *IF THIS SN IS NOT SAVED BY PREVIOUS CYCLE AND NO INWARD POINTERS
   *THEN
      *REMOVE THIS SN FROM CHAINS
      *CHANGE THIS SN TO AVAILABLE STATUS
*ELSE
   *FIND SN PARENT
   *IF SN IS ALREADY A CHILD
   *THEN
      *CREATE NEW SN AS A NEW CYCLE
      *IF ATTRIBUTE NAMES DO NOT MATCH
      *THEN
         *SET @F.DBE AND @L.DBE TO NULL AND #DBE = #INWARDS = 0 IN NEW NODE
         *SCAN DBES
            *PERFORM DELETE FUNCTION
         *SET @F.LLH AND @L.LLH TO NULL AND #LLH = 0 IN NEW NODE
         *SCAN LLH NODES
            *PERFORM UNLINK FUNCTION
      *ELSE
         *TRANSFER DBE AND LL FIELDS AND SET # INWARDS TO #DBE +2, 3 or 4
   *ELSE
      *CREATE NEW SN AS A NEW CHILD
   *INITIALIZE NEW SN HEADER EXCEPT #INWARDS, #LLH AND #DBE
   *INITIALIZE @F.SUB AND @L.SUB AS NULL
   *INITIALIZE CURRENT POINTER TO SN'SYS'
   *LOOP #STRUCT = 1, LEVEL #
      *SCAN OWN NODE OF CURRENT SN FOR MATCH WITH NAME OF NEW SN
      *SCAN NEW MEMBER LIST FOR MATCH WITH NAME OF CURRENT SN
         *IF MATCH IS FOUND THEN PLACE ENTRY IN @P.STRUCTI(@S.STRUCTI)
      *IF OWN FIELD PRESENT
      *THEN
         *CREATE OWN NODE
         *SET #STRUCT = LEVEL#+1
         *SET @P.STRUCT(#STRUCT) AND @S.STRUCT(#STRUCT) TO OWN NODE
      *INSERT ATTRIBUTE NAME LIST AND ARRAY DESCRIPTORS INTO NEW SN
      *COMPLETE HEADER (#CELLS)
```

Table 17. DBMS Function Structure

```
SUBROUTINE UNLINK
*IF THERE IS NOT A CURRENT DATA BASE
*THEN
   *PERFORM RETRIEVE FUNCTION ON DEFAULT DBN
   *IF DEFAULT DBN IS NON RETRIEVABLE
   *THEN
      *PERFORM INITIALIZATION FUNCTION ON DEFAULT DBN
*FIND PARENT SN (NEW CURRENT SN)
*FIND LLH
*FLAG LLH FOR DELETION
*SCAN LINKED LIST CHAIN NODES
   *CREATE NEW CYCLE SHOWING LOSS OF LINKED LISTS
   *IF NO INWARD POINTERS TO OLD LLC AND IT IS NOT SAVED BY PREVIOUS CYCLE
   *THEN
      *CHANGE STATUS OF OLD LLC NODE TO AVAILABLE
*IF THIS LLH NOT SAVED BY PREVIOUS CYCLE AND NO INWARD POINTERS
*THEN
   *REMOVE THIS LLH FROM CHAINS
   *CHANGE STATUS OF THIS LLH TO AVAILABLE
```

Table 18. DBMS Function Unlink

```
SUBROUTINE CREATE
*IF THERE IS NOT A CURRENT DBN
*THEN
   *PERFORM RETRIEVE FUNCTION ON DEFAULT DBN
   *IF DEFAULT DBN IS NON RETRIEVABLE
   *THEN
      *PERFORM INITIAL FUNCTION ON DEFAULT DBN
*FIND SN AND CALL IT THE CURRENT SN
*ALLOCATE A NEW DBE AND CALL IT THE CURRENT DBE
*LINK THIS DBE ONTO CHAIN OF DBES ASSOCIATED TO THE CURRENT SN
*PERFORM STORE FUNCTION ON VALUE KEYWORD PACKET IF PRESENT
```

Table 19. DBMS Function Create

```
SUBROUTINE ENTER
*IF THERE IS NOT A CURRENT DBN
*THEN
   *PERFORM RETRIEVE FUNCTION ON DEFAULT DBN
   *IF DEFAULT DBN IS NON RETRIEVABLE
   *THEN
      *PERFORM INITIAL FUNCTION ON DEFAULT DBN
*FIND DBE AND CALL IT CURRENT DBE
*IF NO SET OWNER SPECIFIED (MISSING SET KEYWORD PHRASE)
*THEN
   *ERROR
*ELSE
   *FIND DBE SET OWNER
   *FIND LOCATION IN DBE SET
   *CHAIN THIS DBE INTO SET
```

Table 20. DBMS Function Enter

```
SUBROUTINE FIND
*IF THERE IS NOT A CURRENT DBN
*THEN
   *PERFORM RETRIEVE FUNCTION ON DEFAULT DBN
   *IF DEFAULT DBN IS NON RETRIEVABLE
   *THEN
      *PERFORM INITIAL FUNCTION ON DEFAULT DBN
*FIND DBE AND CALL IT THE CURRENT DBE
```

Table 21. DBMS Function Find

```
SUBROUTINE RETURN
*IF THERE IS NOT A CURRENT DBN
*THEN
   *PERFORM RETRIEVE FUNCTION ON DEFAULT DBN
   *IF DEFAULT DBN IS NON RETRIEVABLE
   *THEN
      *PERFORM INITIAL FUNCTION ON DEFAULT DBN
*FIND DBE AND CALL IT THE CURRENT DBE
*IF ATTRIBUTE LIST PRESENT (NAMES KEYWORD)
*THEN
   *SCAN ATTRIBUTE LIST PRESENTED
      *FIND THIS ATTRIBUTE
      *IF PRINT SPECIFIED
      *THEN
         *PRINT WITH FORMAT CONSISTENT WITH ATTRIBUTE VALUE TYPE
      *IF CORE SPECIFIED
         *VERIFY ENOUGH CORE PROVIDED
         *TRANSFER ATTRIBUTE NAMES AND VALUES
      *IF A FILE IS SPECIFIED
         *TRANSFER ATTRIBUTE NAMES AND VALUES
*ELSE
   *SCAN LIST OF ATTRIBUTE ON SN ASSOCIATED WITH THE CURRENT DBE
      *FIND THIS ATTRIBUTE
      *OUTPUT AS ABOVE
```

Table 22. DBMS Function Return

```
SUBROUTINE DELETE
*IF THERE IS NOT A CURRENT DBN
*THEN
   *PERFORM RETRIEVE FUNCTION ON DEFAULT DBN
   *IF DEFAULT DBN IS NON RETRIEVABLE
   *THEN
      *PERFORM INITIAL FUNCTION ON DEFAULT DBN
*FIND DBE AND CALL IT THE CURRENT DBE
*FLAG DBE FOR DELETION
*IF THIS DBE DOES NOT NEED TO BE SAVED AS A PREVIOUS CYCLE
*THEN
   *REMOVE THIS DBE FROM CHAINS AND ASSOCIATED (SETS, LINKS, DBE)
   *CHANGE STATUS THIS DBE TO AVAILABLE
*ELSE
   *LEAVE DBE AS ARCHIVED COPY
```

Table 23. DBMS Function Delete

```
SUBROUTINE REMOVE
*PERFORM RETURN FUNCTION WITH SPECIFIED PARAMETERS
*PERFORM DELETE FUNCTION FOR THIS DBE
```

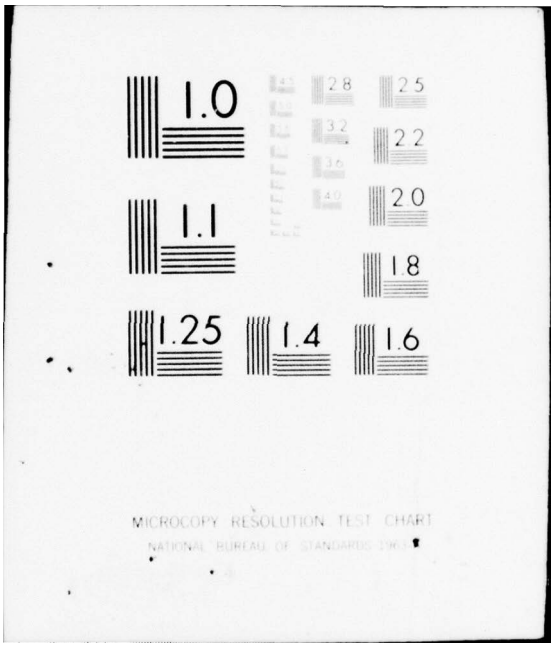
Table 24. DBMS Function Remove

AD-A034 228 BDM CORP EL PASO TEX F/G 9/2
DATA BASE MANAGEMENT SYSTEM REFERENCE MANUAL.(U)
OCT 76 J M PHELAN F29601-74-C-0017
UNCLASSIFIED AFWL-TR-75-159 NL

2 OF 2
AD
A034228

END

DATE
FILMED
2-77


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
SUBROUTINE STORE
*IF THERE IS NOT A CURRENT DBN
*THEN
   *PERFORM RETRIEVE FUNCTION ON DEFAULT DBN
   *IF DEFAULT DBN IS NON RETRIEVABLE
   *THEN
      *PERFORM INITIAL FUNCTION ON DEFAULT DBN
*FIND DBE AND CALL IT THE CURRENT DBE
*GENERATE NEW CYCLE OF CURRENT DBE ALLOWING SPACE FOR NEW VALE ENTRIES
*DELETE PREVIOUS COPY IF POSSIBLE
*SCAN LIST OF ATTRIBUTE VALUE ASSIGNMENTS IF PRESENT
   *IF MULTI CELL VALUE
   *THEN
      *GENERATE VALUE NODE
      *LINK TO DBE DATA NODE
   *ELSE
      *SET VALUE IN DBE DATA NODE
*SCAN PARENT SN'S
   *SCAN LLH THIS SN IF PRESENT (IGNORE FIRST LLH)
      *IF THIS DBE QUALIFIES (CLASS)
      *THEN
         *IF FIRST ENTRY IN LLC
         *THEN
            *CREATE A NEW LLC
            *LINK TO DBE
         *FIND LOCATION IN LINK LIST FOR THIS DBE
         *CHAIN THIS DBE ONTO THE LINK LIST
```

Table 25. DBMS Function Store

```
SUBROUTINE UNFILE
*IF THERE IS NOT A CURRENT DBN
*THEN
   *PERFORM RETRIEVE FUNCTION ON DEFAULT DBN
   *IF DEFAULT DBN IS NON RETRIEVABLE
   *THEN
      *PERFORM INITIAL FUNCTION ON DEFAULT DBN
*FIND DBE AND CALL IT THE CURRENT DBE (ORDER KEYWORD)
*UNCHAIN THIS DBE FROM SET
```

Table 26. DBMS Function Unfile

```
SUBROUTINE RAGGEN

*IF RAGGED TABLE DOES NOT ALREADY EXIST
*THEN
   *IF FIRST SPECIFICATION IS A LEAF
   *THEN
      *PRINT WARNING (SINGLE LEAF RTNS ARE NOT EFFICIENT)
      *ALLOCATE INCREMENT OF RTN STORAGE
      *GENERATE RTN OF ONE LEAF
   *ELSE
      *ALLOCATE INCREMENT OF RTN STORAGE
      *INITIALIZE RTN HEADER
      *GENERATE (#BRANCHLETS) CELLS OF UNDEFINED CONTINUATIONS (LEAVES)
*ELSE
   *IF CYCLE OF RTN = CURRENT CYCLE
   *THEN
      *CREATE NEW CYCLE OF RTN
   *FOLLOW TREE TO NEW NODE
   *IF NEW NODE IS TO BE A BRANCH
   *THEN
      *IF NODE ENTRY IS NOT UNDEFINED CONTINUATION
      *THEN
         *PRUNE OLD BRANCH (LEAF)WHILE INSERTING UNDEFINED CONTINUATION NODE
         *IF OLD NODE WAS A BRANCH THEN COMPACT NEW RTN
      *IF NODE ENTRY IS NOT THE LAST SUBCELL OF THE RTN
      *THEN
         *IF NOT ENOUGH SPACE FOR 3 *#BRANCHLETS
         *THEN
            *IF EFFICIENCY THIS CHUNK IS HIGH
            *THEN
               *ALLOCATE INCREMENT OF RTN STORAGE
               *IF NOT SEQUENTIAL PAGE THEN CHAIN AS PERM EXTENSION TO RTN
            *ELSE
               *COMPACT
         *ELSE
            *CHAIN DEFINED TEMPORARY CONTINUATION
         *PLACE IN (#BRANCHLETS) CELLS OF UNDEFINED CONTINUATION
      *ELSE
         *IF NOT ENOUGH SPACE FOR 3*(#BRANCHLETS-1)
         *THEN
            *IF EFFICIENCY THIS CHUNK IS HIGH
            *THEN
               *ALLOCATE INCREMENT OF RTN STORAGE
               *IF NOT A SEQUENTIAL PAGE THEN CHAIN AS PERM EXTENSION TO RTN
            *ELSE
               *COMPACT
            *PLACE IN (#BRANCHLETS) CELLS OF UNDEFINED CONTINUATION
   *ELSE (NODE ENTRY IS THE LAST SUBCELL OF THE RTN)
```

Table 27. DBMS Ragged Table Generation

```
      *IF NODE ENTRY IS NOT UNDEFINED CONTINUATION
      *THEN
         *PRUNE OLD BRANCH(LEAF) WHILE INSERTING UNDEFINED CONTINUATION
         *IF OLD NODE WAS A BRANCH THEN COMPACT NODE
      *ELSE
         *INSERT NEW LEAF
   *IF # UNDEFINED CONTINUATIONS IS ZERO
   *THEN
      *COMPACT RTN
   *ELSE
      *COMPUTE EFFICENCY AS (1 - 3 (#* - #?)/#Cells)
      *IF # OF DEFINED CONTINUATIONS EXCEED MAXIMUM OR EFFICIENCY LESS THAN MINIMUM
      *THEN
         *COMPACT ALL RTN STORAGE INCREMENTS
*RETURN
```

Table 27. DBMS Ragged Table Generation (Concluded)

```
SUBROUTINE COMPACT RTN

*NOTE COMPACTS ONLY ONE INCREMENT OF STORAGE
*IF THERE ARE DEFINED CONTINUATIONS OR PERM CONTINUATIONS
*THEN
   *ALLOCATE NEW INCREMENT OF RTN STORAGE
   *PREORDER SCAN RTN KEEPING LEVEL POINTERS
      *IF DEFINED CONTINUATION
      *THEN
         *FOLLOW LINK
         *CALCULATE PTR OFFSET THIS LEVEL
      *TRANSFER NODE AND BIAS PTRS
      *IF PERM CONTINUATION
      *THEN
         *REMEMBER FIRST TWO
   *SCAN TABLE OF TWO PERM CONTINUATIONS (IF PRESENT)
      *TRANSFER NODES UNTIL ORIGINAL STORAGE INCREMENT FULL AS DEFINED CONTINUATION
*RETURN
```

Table 28. DBMS Ragged Table Compaction

APPENDIX A

DBMS COMMANDS AND CARD FORMATS

This appendix discusses the DBMS commands and card formats in detail.

Input cards to DBMS are free form in structure. Punching on a card may begin in any arbitrary column for any command. The blank may be used as an optional character which is ignored except when it occurs in an alpha string or the text of a comment. Continuation from one card to the next is accomplished whenever any of the characters +-,/*(=. or $ appear as the last non-blank character on the card to be continued (this rule does not apply to comment cards). Only the first 72 columns of the card may be used for keypunching commands.

Each command must start on a new card. Commands are executed in the order in which they are encountered in the input stream.

Comment cards may be included in the input deck. They always start with the $ character and cannot be continued from one card to the next. As many comment cards as desired may be included in the deck and they may be arbitrarily inserted (except they may not be inserted in the middle of a sequence of card continuations). Imbedded blanks in comment cards are preserved.

All commands are of the form:

FUNCTION, PARAMETER LIST

The functions are outlined in Structure Level 1. A functional description of each function is presented in Structure Level 3. The functional description is intended for the interested reader and should not be considered a necessity for using DBMS.

The parameter list is in the form of a keyword followed by necessary parameters. Each keyword plus its parameters is called a field. If more than one field is present they must be separated by commas. The order of fields is immaterial except as mentioned in the case of protection keys and attribute values. Items which are optional are shown in brackets. Items in nested brackets may appear only if options in outer brackets are chosen. Keywords are capitalized and default optional values are either underlined or mentioned in the following text. If one word is to be chosen from a group of words, the possible words are listed in a column and enclosed in parentheses. Fields and parts of fields will be enclosed by a single quote (') when they appear as part of the following text, except where obvious. When encoding a command all lists must be enclosed by parentheses.

I) Type 1 Functions

Type 1 functions are all utilities. Besides the tasks mentioned below Type 1 functions identify the name of the current data base. If a 'DBN = name' (or 'FROM = name' on a COPY) field appears then the file called name is the new data base. Note that the default data base name is DB. If another data base is current prior to the application of a utility function, that data base is cleaned up and all DBMS internal references to the old file are converted to the new file (e.g., see INITIAL).

The file names used must satisfy the local restrictions on file naming conventions. When DBMS is used under CDC Scope 3.4, a catalogued file which is not currently attached will be dynamically attached while it is required for the DBMS and then returned to the system. In order to dynamically attach a catalogued file the subfield SYSKEY must contain the required system information for the file. On many systems fewer characters are allowed for local than for catalogued file names. The local file name used while dynamically attaching a catalogued file will be the first n characters of the catalogued file name, where n is the maximum number of characters allowed for a local file name. If the name given is neither a local nor a catalogued file name, DBMS will attempt to attach a temporary file with the name given as a local file name. Note that a dynamically assigned catalogued file is returned after use but a dynamically assigned temporary file is retained. Under all operating systems a file may be assigned before DBMS is called.

As mentioned the SYSKEY field is an alpha string which defines the necessary system information to dynamically attach a catalogued file. To encode an alpha string, choose a delimiter. The delimiter is placed immediately before and after the alpha text. Anywhere in the text the delimiter appears it must be replaced by two delimiters. Examples of alpha strings are presented with the Type 3 function CREATE.

For each file referenced in a utility function, the DBMS determines if the file device is tape or random access. If random access, the DBMS further determines if the file is already a data base.

The parameters 'rkey', 'wkey', and 'akey' are order dependent and any combination of the three may be present. If 'rkey' or 'wkey' are absent, the delimiter '/' must still be included. The parameter 'rkey' represents a request for permission to retrieve (read) data currently stored in the data base. The parameter 'wkey' represents a request for permission to store new data into a data base. The parameter 'akey' represents a request to alter existing data in a data base. These keys are not intended to provide absolute security of data bases but rather reduce the possibility of accidental destruction of a data base. These keys may be placed into the data base only with the Type 1 function INITIAL. If the field 'DBN = name' is absent, the parameters 'rkey', 'wkey', and 'akey' are ignored.

The DBMS utility functions are:

A) COMPRESS a data base

COMPRESS {,DBN = name} {,KEY = rkey/wkey/akey} {,SYSKEY = syskey}

Following the procedure mentioned in the general discussion of Type 1 functions, the file 'name' is attached if necessary.

The result of a successful COMPRESS function is to remove unused space in a data base. This unused space is the result of scratch areas used by DBMS while building nodes. Although DBMS attempts to use this space, some fragmentation does occur (see also function CYCLE below).

B) COPY a data base from one file to another

$$\text{COPY dbn fields ,TO = name } \left\{,\text{LINKS} = \left(\frac{\text{SAVE}}{\text{DELETE}}\right)\right\}\left\{,\left(\frac{\text{INCLUDE}}{\text{EXCLUDE}}\right) = (\text{snlist})\right\}$$

where the dbn fields are

{,FROM = name {(cycle number)}}{,KEY = rkey/wkey/akey} {,SYSKEY = syskey}

The default parameter list for a COPY card is 'FROM = current data base, TO = current data base, LINKS = SAVE'. The parameters for the FROM and TO fields are the names of local or catalogued files. Following the procedure mentioned in the general discussion of Type 1 functions, the files are attached if necessary. The SYSKEY subfields must be the same for both files if they are dynamically attached.

The FROM file is verified to be a data base with matching 'rkey' field. The 'wkey' and 'akey' are used to grant write and alter permissions of the FROM data base. The DBMS determines if the TO file is tape or direct access. If the TO file is a direct access and already a data base, the 'wkey' and 'akey' must match those given on the DBMS control card.

The result of a successful COPY function is to move all or part of a data base from one file to another. If the TO file is a tape, then the data base is written in a standard format allowing for easy portability (see the Type 1 function RESTORE). If a cycle number is given for the FROM data base, only that cycle is moved. If the FROM name subfield is the same as the TO name subfield only the cycle number specified (default is the current cycle) is saved and all others are discarded. If a cycle number is not specified and the TO name subfield is different from the FROM name subfield, all cycles are copied.

If LINKS = SAVE is specified, the link list information is copied. If the INCLUDE/EXCLUDE subfield is omitted all SNs and DBEs are copied. The form of snlist is a list of SNs separated by commas and enclosed in parentheses. If an INCLUDE subfield is present, only those SNs mentioned and

their ancestors are copied. If an EXCLUDE subfield is present, the SNs mentioned and their decendents will not be copied. The only DBEs that will be copied are those associated with SNs that are copied.

C) Increment the CYCLE number of a data base (See structure level ?)

CYCLE {,DBN = name} {,KEY = rkey/wkey/akey} {,SYSKEY = syskey}

Following the procedure mentioned in the general discussion of Type 1 functions, the file 'name' is attached if necessary.

The result of a successful CYCLE function is to archive a copy of the data base. Any future alterations to the data base will result in a copy of the affected nodes to be made and alterations made to the copies. The old nodes retain their information intact and can be retrieved by requesting the proper cycle number. Where efficiency is of importance, it should be noted that only two words in the data base are changed as a result of a CYCLE function. When a node is altered (causing a new cycle) a copy of the node is made, but all pointers to the node are left pointing to the old cycle of the node. When a reference is made to an out of date node through a pointer, the pointer is updated. As such the cycle information requires more file space, but the other computer resources are kept to a minimum.

It is advisable to copy a data base to tape and set the data base cycle back to zero occasionally.

The maximum cycle number is 4096, at which time the oldest cycle is lost. The number of cycles saved may be reduced during site initialization of DBMS.

D) DISPLAY a segment of a data base

DISPLAY dbn fields { , (SN = sn ,LISTS / BELOW = sn / ABOVE = sn / DBE = dbe) }

where the dbn fields are:

{,DBN = name} {,KEY = rkey/wkey/akey} {,SYSKEY = syskey}

The default parameter list for a display card is 'DBN = current data base, DBE = SYS*SYS'. Following the procedure mentioned in the general discussion of Type 1 functions, the file 'name' is attached if necessary.

The result of a successful DISPLAY function is to output information describing a specified SN or DBE. Note that both SNs and DBEs may not be specified on the same card. In either case, one node is specified (SN 'sn' or DBE 'dbe').

A single SN can be selected with the option 'SN = sn' or a group of SNs can be selected with the options BELOW, ABOVE, or LISTS. The SNs on the subtree with root SN 'sn' can be selected with the option 'BELOW = sn'. The ancestral SNs of a sn can be selected with the option 'ABOVE = sn'. Pertinent information about the linked lists attached to SN 'sn' can be output with the option 'SN = sn, LISTS'.

The format of the information output depends on the OUTPUT medium (CORE, FILE or PRINT), the number of nodes selected, and the kind of nodes selected.

1) If PRINT is specified the following information is printed for each of the options listed below:

   a) 'SN = sn':

      i) Each attribute name and value type by retained cycle,

      ii) The names of the SN's children,

      iii) The number of linked lists and DBEs for this SN,

      iv) If the LISTS option is chosen, the class attributes (which DBE may belong to the list) and the order ranking definition (See Type 2 function STRUCTURE).

   b) 'BELOW = sn', or 'ABOVE = sn', for each SN:

      i) The attribute names and value type for current cycle,

      ii) SN name and SN tree level number,

      iii) The number of link lists and DBEs for this SN.

   c) 'DBE = dbe':

      i) Associated SN's generic name (i.e., SYS., etc)

      ii) Each attribute name and value by retained cycle number (except values in ragged tables, arrays and alpha fields containing more than 50 characters),

      iii) Set ownership (yes or no),

      iv) Levels of DBEs which own this DBE.

E) INITIALIZE a data base

```
INITIAL {,DBN = name} {,KEY = rkey/wkey/akey}  {,SYSKEY = syskey}
```

The default parameter list for an INITIAL card is DBN = DB. The parameter 'name' is the local or catalogued name of the direct access file on which the data base is to reside. Following the procedure mentioned in the general discussion of Type 1 functions, the file is attached if necessary. If the file was a data base prior to this call, the values of rkey, wkey, and akey (read, write, and alter keys) on the control card must match the values stored in the data base. If a match fails the program executes an error exit.

The result of a successful INITIAL function is to generate a data base header (Structure Level 2.7), the SN 'SYS', the DBE 'SYS', and then initialize DBMS variables within DBE 'SYS' including rkey, wkey, and akey (See Structure Levels 2.3 and 2.5).

F) LIST out the date and time of all cycles

```
LIST  {,DBN = name}  {,KEY = rkey/wkey/dkey}  {,SYSKEY = syskey}
      {,(CORE = location(size)
         FILE = fname
         PRINT)}
```

The default parameter list for a LIST card is 'DBN = current data base name, PRINT'. Following the procedure mentioned in the general discussion of Type 1 functions, the file 'name' is attached if necessary.

The result of a successful LIST function is that the cycle date and time information is transferred from the data base to the specified device. If PRINT is specified, the information is printed. If 'FILE = fname' is specified, the file called 'fname' is attached and the information written to it sequentially (See TYPE 1 function DISPLAY for format).

If 'CORE = location(size)' is specified the information is transferred to the core location given (See Type 1 function DISPLAY for format). The options FILE and CORE are intended to be used with short commands for interprogram communications (See Structure Level 1.0)

G) RESTORE a data base from tape to random access

```
RESTORE  {,TAPE = name} {,DBN = name}{,KEY = rkey/wkey/akey}
         {,SYSKEY = syskey}
```

The default parameter list for a RESTORE card is 'TAPE = TAPE, DBN = DB'. The parameters for the FROM and TO fields are the names of local or catalogued files. Following the procedure mentioned in the general discussion of Type 1 functions, the files are attached if necessary. The SYSKEY subfields must be the same for both files if they are dynamically attached. The TAPE file is verified to be a tape and the DBN file is verified to be a random access file.

The result of a successful RESTORE function is to copy a data base from a tape to a random access device. The tape must have been written by the DBMS function COPY. However, the machine used during the DBMS copy operation may be of a different brand.

H) RETRIEVE an existing data base

RETRIEVE {,DBN = name}{,KEY = rkey/wkey/akey}{,SYSKEY = syskey}

The default parameter list for a RETRIEVE card is 'DBN = DB'. The parameter 'name' is the local or catalogued name of the direct access file on which the data base resides. Following the procedure mentioned in the general discussion of Type 1 functions, the file 'name' is attached if necessary. The file must be a data base or the job is put into an error mode. The values of rkey, wkey, and akey are compared and the appropriate permissions are granted. A permission violation (e.g. trying to alter an existing data base node when alter permission not granted) will cause the job to be put into an error mode.

The result of a successful retrieve function is to verify that the data base has at least a minimum structure. In particular the header, the SN 'SYS', the DBE 'SYS' and the internal variables are checked.

II) Type 2 Functions

Type 2 functions deal with the generation and control of a data base's structure. The data base is always the current data base name (See general discussion of Type 1 function in this appendix). The two types of nodes affected are SNs and link lists (LLs) (See Structure Levels 1.2 and 1.5). In each case a required parameter field is 'SN = sn' or 'LL = sn' where sn is the full name of a SN (See Structure Level 1.3).

The DBMS structure node functions are:

A) LINK a group of DBEs into an ordered list

LINK ,LL = sn $HLn {,CLASS = (expr)} {,RANK = (orderlist)}

LINK attaches a link list header (LLH) called Ln to SN 'sn'. The names of linked lists attached to a SN must be unique. Link lists are internally

numbered sequentially starting with 1 for the oldest, 2 for the next to oldest, etc. The LLH contains information describing which DBEs are in the list (CLASS) and what determines the ordering (RANK). (See Structure Level 1.5). The membership is defined by the parameter 'expr'. The ordering algorithm is defined by the 'orderlist' parameter and is interpreted as for the TYPE 2 function STRUCTURE. The form of 'expr' is the same as an ANSI FORTRAN logical expression with the relational expression redefined. The form of the DBMS relational expression is defined in Structure Level 1.3.

The field orderlist specifies how sets owned by any DBE associated with the SN being defined are ordered. The format of an orderlist is a series of order elements separated by commas and the list must be enclosed in parentheses. The form of an order element is:

$$\left(\begin{pmatrix}\text{HIGH}\\ \text{LOW}\end{pmatrix} \quad \text{attribute name (dimension)} \right)$$
$$\text{LIFO}$$
$$\underline{\text{FIFO}}$$

where dimension is used to specify an array or ragged table element. The order list is position dependent and is interpreted as:

order by (order element 1) and settle ties by
(order element 2) and settle remaining ties by
(order element 3) and so on

If a set member does not have an attribute required by an order element it is considered last for that order element. If a tie still exists after all order elements have been considered, the tie is settled by FIFO ordering.

The defaults for 'CLASS = (expr)' and 'RANK = orderlist' is for all the eligible DBEs to be linked into a FIFO queue (See Structure Level 1.5).

B) Add or modify the data base structure

$$\text{STRUCTURE } \{,\text{SN = sn}\} \quad \left\{\begin{pmatrix}\text{args}\\ ,\text{DELETE}\end{pmatrix}\right\}$$

where args are:

{,ATTRIBUTE = (alist)}{,MEMBER = (mlist)} {,OWN = ownlist)} {,RANK = orderlist}

Since the STRUCTURE function deals with the heart of the data base organization, there are no defaults allowed. If the DELETE option is chosen, the SN 'sn' and all of its descendents plus their associated DBEs are flagged as deleted as of the current cycle. Previous cycles of this node can be retrieved by copying that particular cycle of the current data base to a new data base (See COPY function).

If the DELETE option is not selected a new SN is created (See Structure Level 3.3). The parameter alist describes the names and value type of attributes of the DBE associated with SN 'sn'. The format of alist is a series of attribute definitions separated by commas (,). Note alist must be enclosed in parentheses.

The form of an attribute definition is:

```
(INTEGER name (dimensions)
 REAL name (dimensions)
 DOUBLE name (dimensions)
 COMPLEX name (dimensions)
 ALPHA name (dimensions)
 BITS(width)name (dimensions)
 DB name
 RAGGED name)
```

where name is a single name or a list of names separated by commas and enclosed in parentheses. An attribute name may be from 1 to 9 alphanumeric characters starting with a letter. Width is the number of bits per value (default is 52) and 'dimensions' may be from one to three dimensions, in ANSI FORTRAN format. If the dimension is (1), (1,1) or (1,1,1) it is assumed the dimensionality will be specified by a Type 3 function. In any case the dimensionality of an array may be altered by a Type 3 function (see Type 3 functions SET and CREATE).

The member list 'mlist' and the own list 'ownlist' contain SNs and define which DBEs may belong to a given set. In particular a DBE associated with the SN being defined may belong to any set attached to a DBE which is associated with any SN in the member list. Further a DBE associated with the SN being defined may own as a set any DBE associated with a SN in the own list. Thus, there are two ways to define set membership and either or both may be used. The parent of the SN being defined is always in the member list by default. For details of set membership see Structure Level 1.2. The format for both 'mlist' and 'ownlist' is a list of SNs separated by commas (,) and the list must be enclosed in parentheses.

The meaning of orderlist is explained in the discussion of the Type 2 function LINK.

(C) UNLINK a linked list of DBEs

```
UNLINK, LL = SN$Hln
```

See Type 2 function LINK for the meaning of SN$Hln. The specified LL is flagged for deletion. The link list will be available as long as previous cycles are.

III) Type 3 Functions

Type 3 functions deal with the generation and control of data entries (DBE). Each Type 3 function except CREATE has an optional DBE name parameter field. The data base maintains the identity of the most recently used DBE. Thus the same DBE can be repeatedly accessed without specifying its name each time. Further the Type 3 function FIND does nothing more than set the most recently used DBE pointer (current DBE) to a particular DBE. Thus except for CREATE, the discussion below assumes the most recent DBE is being accessed unless the 'DBE = name' field appears.

The Type 3 functions are:

A) CREATE a DBE using a SN as a template

$$\text{CREATE, SN = sn } \left\{ \text{,VALUE = (vpairs)} \right\}$$

The CREATE function uses the template information from SN 'sn' to create a DBE. All the attributes, except those mentioned in the VALUE field, are flagged as undefined. The format of the field vpairs is a series of attribute name/value pairs separated by commas and enclosed in parentheses. The attribute name/value pairs are of the form 'attribute name = value'. The form of value must agree with the attribute value type (See Structure Level 1.0). The attribute value types are described below:

i) Alpha string of length smaller than 649 characters.

To encode an alpha text into an alpha string, choose any delimiter from the set ')', '*', '/', ',', '='. The delimter is placed immediately before and after the alpha text string. Anywhere in the text the delimiter appears it must be replaced by two delimiters. As an example the text 'DOG*CAT' can be represented by any of the following:

```
/DOG*CAT/
*DOG**CAT*
```

The following representations are in error:

```
DDDOG**CATD
CDOG*CCCATC
*DOG*CAT*
/DOG*CAT*
```

ii) Real constants are represented in the same fashion as ANSI FORTRAN. Note when transferring from CDC 6600/6700 to IBM 360/370 or UNIVAC 1108/1110 all single precision variables are converted to double.

iii) Integer constants may be represented as binary, octal, hex or decimal numbers by preceding the number with '$BIN', '$OCT', '$HEX' or '$DEC' respectively. Thus the number 65 may be represented as:

```
$BIN 1000001
$OCT 101
$HEX 41
$DEC 65
     65
```

Note that the default is '$DEC'. Truncations caused by transferring a data base from one system to another will be flagged.

iv) COMPLEX constants are represented in the same fashion as ANSI FORTRAN. Loss of precision caused by transferring a data base from one system to another will be flagged.

v) DOUBLE PRECISION constants are represented in the same fashion as ANSI FORTRAN. Loss of precision caused by transferring a data base from one system to another will be flagged.

vii) A BIT STRING constant may be represented in the same fashion as INTEGER constants.

viii) Data base descriptors are in the form of

$$\$Dname\ \left\{\left(\left\{syskey\right\}\quad\left\{\$rkey/wkey/akey\right\}\right)\right\}$$

These fields are identical to those mentioned in the Type 1 function INITIAL.

The form of value may be a simple constant described above or the words $FILE(name) or $CORE (location/length). These fields are the same as the Type 1 function DISPLAY. The format must be in the form described in Structure Level 2.2.

B) ENTER a DBE into a SET

$$ENTER\ \left\{,DBE = dbe\right\}\quad\left\{,ORDER = \begin{pmatrix} FIRST \\ LAST \\ BEFORE\ dbe1 \\ AFTER\ dbe1 \end{pmatrix}\right\}\quad ,\ SET = db2$$

The current DBE is filed into the set owned by DBE 'dbe2'. The normal ordering for this set was defined in the Type 3 STRUCTURE function for the SN associated with DBE 'dbe2'. This ordering can be overridden with the ORDER parameter field. Here FIRST and LAST refer to the first and last DBE in the set, and BEFORE and AFTER refer to DBE 'dbe1'.

C) FIND a specific DBE

FIND ,DBE = dbe

This function is provided as a convenience and merely sets the internal pointer 'current DBE' to DBE 'dbe' (See general discussion of Type 3 functions).

D) RETURN, DELETE, or REMOVE a data node and/or any of its attributes

$$\begin{pmatrix}\text{RETURN}\\ \text{DELETE}\\ \text{REMOVE}\end{pmatrix}\left\{,\text{DBE} = \text{dbe}\right\}\left\{,\text{NAMES} = (\text{alist})\right\}\left\{,\begin{pmatrix}\text{CORE} = \text{location(size)}\\ \text{FILE} = \text{fname}\\ \underline{\text{PRINT}}\end{pmatrix}\right\}$$

RETURN will return the values of specific attributes, DELETE will delete the specified DBE and REMOVE will return the attribute values then delete the DBE. The attribute names are listed in alist and are separated by commas. They are sent to CORE, FILE, or PRINT. If PRINT is chosen then the names of single elements and arrays are listed along with their values. Ragged tables are listed in the format displayed in Structure Level 1.4.

The fields CORE and FILE have the same meaning as for the Type 1 function DISPLAY. The form of the returned attribute values starts with a cell of the format:

ATTRIBUTE NAME	VALUE TYPE

which is cell Type 6 in Structure Level 2.1. This cell is followed by a copy of the information stored.

E) STORE the value(s) of an attribute(s) in a DBE

STORE {,DBE = dbe}{,VALUE = vpairs}

where vpairs is defined as for the Type 3 function CREATE.

F) UNFILE a DBE from a set

UNFILE {,DBE = dbe} , SET = dbe3

The DBE mentioned in the DBE parameter field (current DBE) is removed from the set owned by DBE 'dbe3'. If the current DBE is not a member of the set owned by DBE 'dbe3', an error will result. The options in the DBE parameter field are the same as for the LINK function.

IV) Type 4 Functions

These two functions control the STATUS of the DBMS.

A) END DBMS's manipulation of a DATA base

END DATA

The function END DATA will clean up a data base and negate all internal DBMS pointers to the data base. This is an optional function as the same effect can be obtained with any Type 1 function (this function also occurs automatically when the last command has been processed).

B) Change the MODE of the DBMS

$$\text{MODE} \left\{ , \left(\begin{array}{l} \underline{\text{LONG}} \\ \text{SHORT} \end{array} \right) \right\} \quad \left\{ , \left(\begin{array}{l} \text{CORE} = \text{location(size)} \\ \text{FILE} = \text{fname} \\ \underline{\text{CARD}} \end{array} \right) \right\}$$

The result of a MODE function is to set the input format from alphanumeric to numeric short (See Structure Level 1.0). The input device for the numeric short can be CORE, FILE or CARD. These parameters have the same meaning as for Type 1 function DISPLAY, and indicate where future commands will come from.

APPENDIX B

NODE NAME AND MOTION PHRASE CONVENTIONS

This appendix is intended to be a quick reference for node naming conventions. The following paragraph represents a condensation of Structure Level 1.3 (with motion in links and ancestorial SNs added). It is followed by a table of the node naming conventions.

A full node name begins with 'SYS' and is followed by motion phrases (a motion operator and required parameters). Each motion phrase is interpreted as moving from a starting node to a target node in the data base. The motion phrases are interpreted from left to right. Thus 'SYS' is the starting node for the first motion phrase, and the target node of the first motion phrase is the starting node for the second motion phrase. Table 29 and its footnotes define the target node given the motion phrase and the type of starting node (SN or DBE). The footnotes in Table 29 contain pertinent information regarding structure sets, DBE sets and link lists (see Structure Level 1).

In the first column of Table 29 are the nine possible motion symbols. The second column indicates which motion symbols require a parameter. The parameters (if required) and resultant target nodes are dependent on the type of starting node. Thus, for a SN as the starting node, the third and fourth columns indicate the type of parameter (if required) and the target node respectively. The third column also presents one or more examples of allowable parameters. SNs are named sn, sn1, sn2, ...; DBEs are named dbe, dbe1, dbe2, ..., and LLs are name LL, LL1, LL2, ... The fifth and sixth columns have the information for the case of a DBE as a starting node.

DBMS retains the most recent parameters for the function keywords 'SN' and 'DBE' as the current SN and current DBE, respectively. Thus when specifying a SN or DBE, if no starting node is given the current SN or current DBE is assumed. As an example, if in Figure 3 the current SN is SN 'SYS.A', then SN 'SYS.A.A' may be referenced in short form by '.A'.

	STARTING NODE IS snl			STARTING NODE IS dbel	
SYMBOL	PARAMETER	PARAMETER TYPE	TARGET NODE	PARAMETER TYPE	TARGET NODE
.	Required	SN 1. snl 2. ((name.NE.*snl*) .AND.(name.NE.* sn2*))	SN with name given in structure set of SN 'snl'	expression[7] 1. ((ohms .LE. 3.4) .AND. (name .EQ. *TRASF*))	DBE which satisfies relational expression given and is in set of DBE 'dbel'.[3]
*	None		First DBE associated with SN 'snl'[1,5]		Not allowed
$F	None		First SN identified in the structure set of SN 'snl' (will be first child of SN 'snl')[1,5]		First DBE in set of DBE 'dbel'.[1]
$L	None		Last SN identified in the structure set of SN 'snl'.[1,5]		Last DBE in set of DBE 'dbel'.[1]
$H	Required	list id[6] 1.LL1	First DBE in a linked list (LLH)	Not allowed	Not allowed
$S	None		SN entered just after SN 'snl' into current structure set.[1,2]		Next DBE in current set or list.[1]
$P	None		SN entered just before SN 'snl' into current structure set.[1,2]		Previous DBE in current set.[1]

Table 29. Motion Phrase Construction

		STARTING NODE IS snl		STARTING NODE IS dbel	
SYMBOL	PARAMETER	PARAMETER TYPE	TARGET NODE	PARAMETER TYPE	TARGET NODE
$B	None		Parent SN of SN 'snl'		SN associated with DBE 'dbel'.
/	Required	SN[1] 1. sn[1] 2. (name.EQ. *SN1*)	SN with name given in current structure set.[1]	Expression[1,2]	DBE which satisfies relational expression given and is in current set or list.[1,3]

Table 29. Motion Phrase Construction (Concluded)

Notes:

1. The symbols '.', '$F', '$L, and '$H' specify entry into a structure set, DBE set, or linked list as specified and called current structure set, current set, or current list respectively. Once entry is made into a set or list the symbols '$P', '$L', and '/' will select nodes in the current set or list. The symbols '.', '$F', '$L', and '$H' specify an exit from the current set or list (if one exists) into a new set or list. The symbols '*' and $(B) will cause an exit from the current set or list if one exists. If a set or list reference is made ('$P', '$S', '/') when there is no current set or list the sibling SNs are considered for SN 'A' and the DBE associated with the same SN are considered for DBE 'B'.

2. Structure sets are ordered as follows: siblings in FIFO order followed by nonsiblings in FIFO order.

3. See Appendix A Type 2 function LINK for definition of relational expression.

4. See node specification format in Structure Level 1.3.

5. The symbol '*' specifies a group of DBE nodes associated with SN 'A'. As such '$F' refers to the first DBE of SN 'A' created and '$L' refers to the last DBE of SN 'A' created.

6. The name of a linked list or the number of a linked list. Linked lists are numbered sequentially starting with 1 for the oldest to N for the newest.

7. See Appendix A, Type 2 function LINK for the definition of relational expression.

APPENDIX C

DBMS EXAMPLES

1. Example 1.

```
$GENERATION OF FIGURE 6
$NOTE, EXCEPT IN ALPHA FIELDS BLANKS ARE IGNORED.
INITIAL,DBN=FIG6
STRUCTURE,SN=SYS.SIMS,
      ATTRIBUTE = (ALPHA CODE, ALPHA ORIGIN)
STRUCTURE,SN=.XTORS,
      ATTRIBUTE = (ALPHA DOMAIN, INTEGER DOMCODE)
STRUCTURE,SN=.MODELS,
      ATTRIBUTE = (ALPHA USAGE, ALPHA COMMENT, ALPHA NAME, ALPHA PERSON,
                   ALPHA PHONE, ALPHA MODEL),
      MEMBER = (SYS.SIMS.XTORS)
CREATE,SN=SYS.SIMS,              VALUE = (CODE=*SIM1*,ORIGIN=*A*)
ENTER,SET=SYS*$F
CREATE,SN=.XTORS,                VALUE = (DOMAIN=/FREQUENCY/,DOMCODE=1)
ENTER,SET=SYS.SIMS*$F
CREATE,                          VALUE = (DOMAIN=/TIME/,DOMCODE=2)
ENTER,SET=SYS.SIMS*$F
CREATE,SN=.MODELS,               VALUE = (USAGE=*SPECIFIC*,COMMENT=*...*,
       NAME=*2N3638*,PHONE=*683-427-9542*,PERSON=*JIM BROWN*,MODEL=*..*)
ENTER,SET=SYS.SIMS.XTORS*$F
CREATE,                          VALUE = (USAGE=*SPECIFIC*,COMMENT=
       *ERERS MULL*,NAME=*2N3638*,PERSON=*JOE SMITH*,PHONE=
       *593-526-8301*,MODEL=*...*)
ENTER,SET=SYS.SIMS.XTORS*$F$S
CREATE,                          VALUE = ( USAGE=*GENERAL*,COMMENT=*...*,
      NAME=*2N3638*,PERSON=*BOB SMITH*,PHONE=*590-566-3021*,MODEL=*...*)
ENTER,SET=SYS.SIMS.XTORS*$F$S
CYCLE
$
$     CYCLE 0 OF THE DATA BASE NOW CORRESPONDS TO THE DBE'S FOR
$     SIMULATION CODE *SIM1* IN FIGURE 6.
CREATE,SN=SYS.SIMS,              VALUE = (CODE=*SIM2*,ORIGIN=*B*)
ENTER,SET=SYS*$F
CREATE,SN=.XTORS,                VALUE = (DOMAIN=/TIME/,DOMCODE=2)
ENTER,SET=SYS.SIMS*$F$S
STRUCTURE,SN=SYS.SIMS.DIODES,
      ATTRIBUTE = (ALPHA GRANULE)
CREATE,                          VALUE=(GRANULE=*FINE*)
ENTER,SET=$B$B*$F$S
$
$     REFERENCING APPENDIX B AND FIGURE 5, THE CURRENT DBE POINTER IS
$     POINTING TO THE ONLY DBE ASSOCIATED WITH SN SYS.SIMS.DIODES  .
$     THUS THE FIRST $B NOTION TAKES US BACK TO SN SYS.SIMS.DIODES, AND
$     SECOND $B TAKES US BACK TO THE SN SYS.SIM  .  THEN THE * TAKES US
$     TO THE DBE ASSOCIATED WITH SN SYS.SIMS , AND THE RELATION FINDS
$     SIM2.
$
```

```
STRUCTURE,SN=.MODELS,          ATTRIBUTE = (ALPHA NAME, INTEGER DOMCODE,
       ALPHA PERSON, ALPHA PHONE, ALPHA MODEL)
CREATE,                        VALUE = (NAME=*1N3636*,DOMCODE=1,
       PERSON=*JOE SMITH*,PHONE=*593-526-8301*,MODEL=*...*)
ENTER,SET=SYS.SIMS.DIODES*$F
CREATE,SN=SYS.SIMS.XTORS.MODELS,
       VALUE = (USAGE=*GENERAL*,COMMENT=*...*,NAME=*2N3638*,
                PERSON=*BOB SMITH*,PHONE=*593-566-8301*,MODEL=*...*)
ENTER,SET=SYS.SIMS.XTORS*$F*S$S
CYCLE
$
$      CYCLE 1 NOW CONTAIN FIGURE 6 ENTIRELY.
$
ENDDATA
**
```

2. Example 2.

```
$
$      THIS EXAMPLE USES THE DATA BASE CREATED BY EXAMPLE 2.
$
RETRIEVE,DBN=SUPER
$
$      THE CURRENT STRUCTURE INFORMATION FOR DBN=SUPER WILL BE PRINTED.
$
$
$
$
$      SYSTEM UNITS DEFINITION.
$
CREATE,SN=SYS.SYSTEM,
       VALUE =(NAME = *LOGIC MODLULE A23.5GN*)
CREATE,SN=SYS.SYSTEM,
       VALUE =(NAME = *MOTOR SERVO SAG.N*)
CREATE,SN=SYS.SYSTEM,
       VALUE =(NAME = *DOPPLER FILTER*)
$
$
$      BOX DEFINITIONS.
$
CREATE,SN=.BOX,
       VALUE = (NAME = *POWER SUPPLY*)
ENTER,SET=SYS.SYSTEM*$F
CREATE,
       VALUE = (NAME=*DRIVER*)
ENTER,SET=SYS.SYSTEM*$F
CREATE,
       VALUE = (NAME=*LOGICS*)
ENTER,SET=SYS.SYSTEM*$F
$
$
$      CIRCUIT DEFINITIONS.
$
CREATE,SN=.CIRCUIT,
```

```
      VALUE = (NAME = *BOARD 1*)
ENTER,SET=$B$B*$F
CREATE,
      VALUE = (NAME = *BOARD 2*)
ENTER,SET=$B$B*$F
CREATE,
      VALUE = (NAME = *BOARD 3*)
ENTER,SET=$B$B*$F
$
$
$     DISCRETE UNIT DEFINITIONS.
$
CREATE,SN=SYS.SYSTEM.BOX.CIRCUIT.DISCRET,
      VALUE =(NUMBER     = *3N3413*          , VENDOR     = *RELIA-TRANS*  ,
              STRUCTURE  = *LAYERED*         , DATA       = *N/A*          ,
              FUNCTION   = *POWER*           , PACKAGE    = *GREEN*        ,
              INPUT      = *VOLTS/AMPS*      , INMODELRR  = *JAKE*         ,
              OUTPUT     = *NOT MUCH*        , OUTMODELF  = *IR3**4*       ,
              OUTMODELR  = *BILL W.*         ,
              INVOLT     = 12.)
$     NOTE THAT THE VALUES FOR OUTCASETR, AND OUTJUNCAP ARE NOT
$     PRESENT.  THIS WOULD CORRESPOND TO THE VALUES NOT BEING KNOWN.
$     THEY CAN BE SPECIFIED LATER OR THEY MIGHT NEVER BE REQUIRED BY
$     THE USER.  IF THEY ARE REFERENCED THEIR ABSENCE WILL BE DETECTED

$     AND FLAGGED BY DBMS.  THE NET RESULT IS THEN DEPENDENT ON DBMP.
$
ENTER,SET=$B$B*$F$S
$
$     TO AVOID REPETITIONS, IT IS ASSUMED THAT MANY ENTRIES ARE MADE
$     INTO THE DATA BASE AND THEY ARE ALL ENTERED IN THE SAME MANNER
$     AS TRANSISTOR 3N3413 ABOVE.
$     NOTE THAT THE DBMP COULD SIMPLIFY THE SET MANIPULATION.  IN
$     PARTICULAR DBMP COULD KEEP TRACK OF ORIGINS FOR INSERTING DBES
$     INTO SETS.  ALSO THE CREATION OF DBES COULD BE AUTOMATED WITH THE
$     HELP OF DBMP.
$
ENDDATA
**
```

3. Example 3.

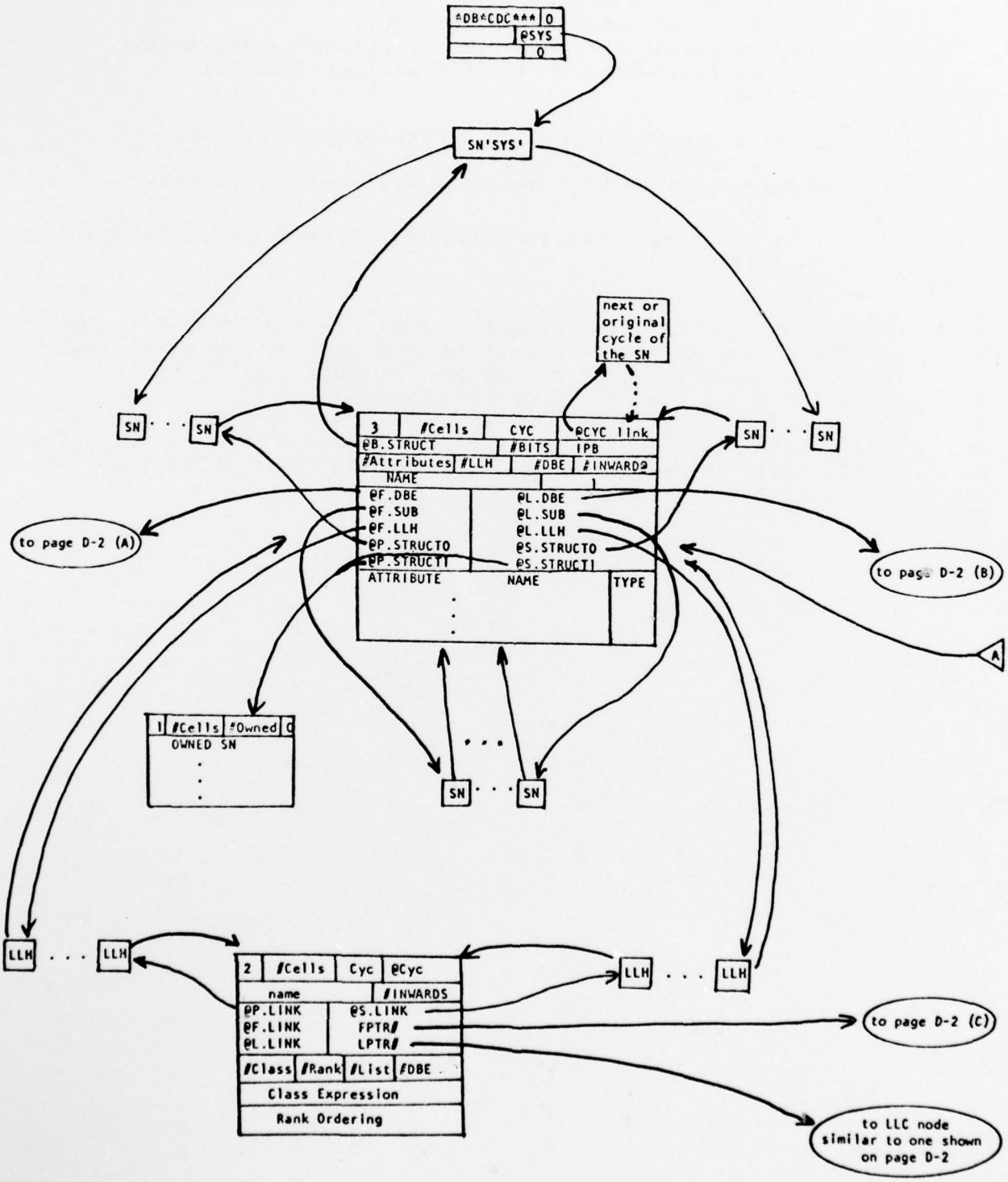
```
$     EXAMPLE 2
INITIAL,DBN=SUPER
STRUCTURE,SN=.SYSTEM,
      ATTRIBUTE=(ALPHA NAME)
STRUCTURE,SN=.BOX,
      ATTRIBUTE=(ALPHA NAME)
$NOTE OWNED BY SYSTEM BY DEFAULT
STRUCTURE,SN=.CIRCUIT,
      ATTRIBUTE=(ALPHA NAME)
STRUCTURE,SN=.DIGITAL,
      ATTRIBUTE=(
            ALPHA(NUMBER,VENDOR,FUNCTION,ACTPROS,PASPROS,PACKAGE,
                  GLASS,PINS,LOGIC,DATA,REFERENCE,INMODEL,OUTMODEL,
                  PWRMODEL,MISMODEL),
             REAL(POWER,OPFREQ,DELAY,RISETIME,NOISEREJ,INVOLT1,
                  INVOLT0,INZ,INSRGZ,INK1,INK2,OUTVOLT,OUTRGZ,OUTK1,
                  OUTK2,PWRVOLT,PWRSRGZ,PWRK1,PWRK2,MISVOLT,MISZ,MISSRGZ,
                  MISK1,MISK2))
STRUCTURE,SN=$B.ELECTRO,
      ATTRIBUTE=(
            ALPHA(NUMBER,VENDOR,FUNCTION,PACKAGE,RATIO,PINS,INPUT,
                  REFERENCE,DATA,FAILMODEL),
             REAL(POWER,OPFREQ,MAXFREQ,MINFREQ,RESIST,INDUCT,CAPAC,
                  VOLTS,CURRENT,ARCVOLTCS,ACRVOLTRF,FAILK1FAILK2))
STRUCTURE,SN=$B.DISCRET,
      ATTRIBUTE=(
            ALPHA(NUMBER,VENDOR,FUNCTION,STRUCTURE,PACKAGE,DATA,
                  REFERENCE,INPUT,INMODELF,INMODELRR,OUTPUT,OUTMODELF,
                  OUTMODELR),
             REAL(POWER,FREQUENCY,RISETIME,DELAYTIME,CURRENT,GAIN,
                  INVOLT,INZ,INZBLK5,INZBLK50,INCASETR,INAMBTR,INJUNCAP,
                  INKF1,INKF2,INKR1,INKR2,OUTVOLT,OUTZBLK5,OUTZBLK50,
                  OUTCASETR,OUTAMBTR,OUTJUNCAP,OUTKF1,OUTKF2,OUTKR1,
                  OUTKR2))
STRUCTURE,SN=$B.LINEAR,
      ATTRIBUTE=(
            ALPHA(NUMBER,VENDOR,FUNCTION,ACTPROS,PASPROS,PACKAGE,
                  GLASS,PINS,DATA,REFERENCE,INMODEL,OUTMODEL,PWRMODEL,
                  MISMODEL),
             REAL(POWER,MAXFREQ,MINFREQ,RISETIME,GAIN,INCMVOLT,
                  INCMZ,INCMSRGZ,INCMRR,INSIGVOLT,INK1,INK2,OUTVOLT,
                  OUTSRGZ,OUTK1,OUTK2,PWRVOLT,PWRSRGZ,PWRK1,PWRK2,
                  MISVOLT,MISSRGZ,MISK1,MISK2))
STRUCTURE,SN=$B.PASSIVE,
      ATTRIBUTE=(
            ALPHA(NUMBER,VENDOR,FUNCTION,VENNUM,PACKAGE,ENCAP,PINS,
                  DATA,ARCMODEL,FAILMODEL),
             REAL(TOLERANCE,RESIST,CAPAC,INDUCT,POWER,VOLT,CURRENT,
                  ARCK1,ARCK2,FAILMODEL,FAILK1,FAILK2))
```

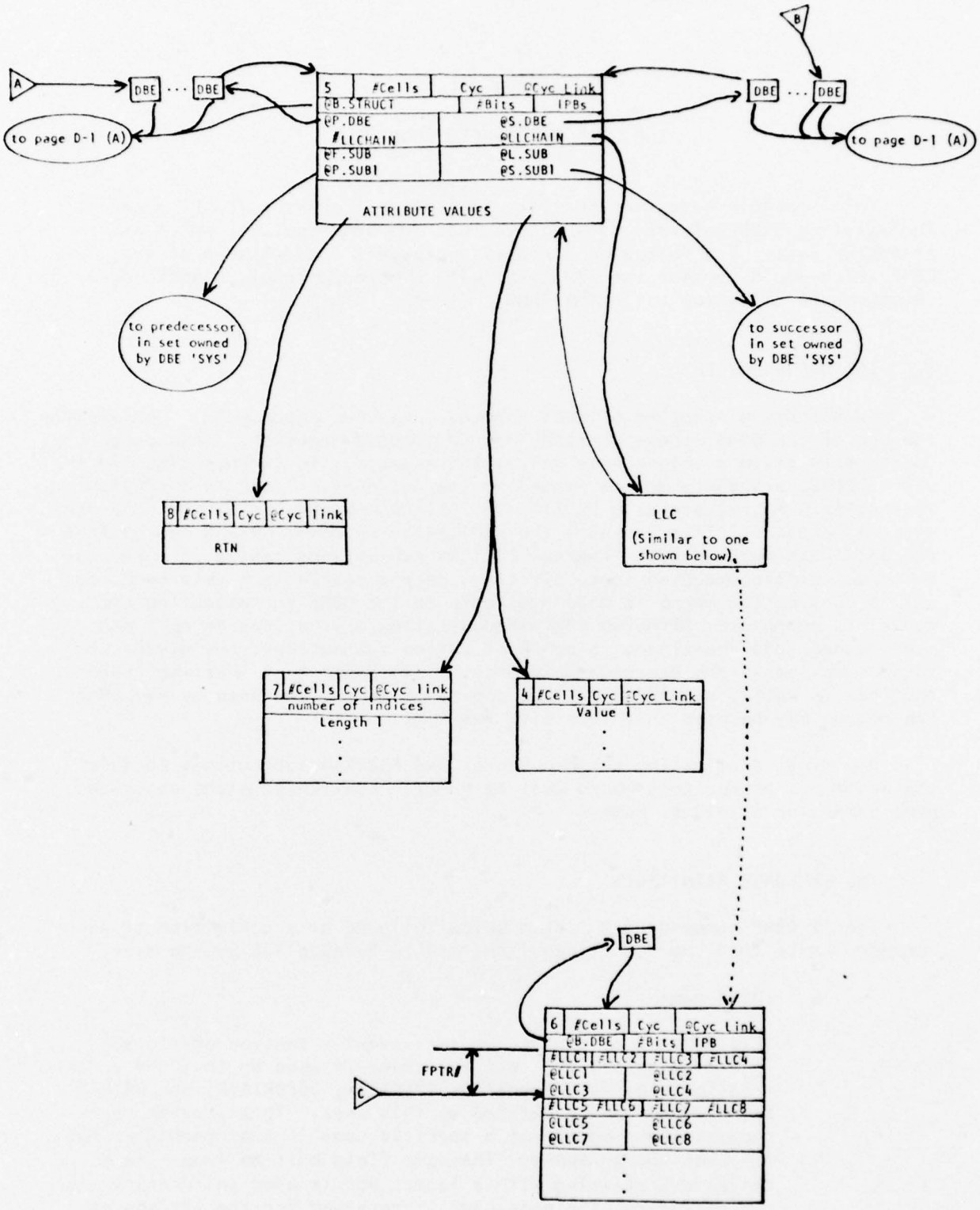
```
STRUCTURE,SN=$B.SUPPRES,
      ATTRIBUTE=(
            ALPHA(NUMBER,VENDOR,FUNCTION,PACKAGE,DATA,BRKMODEL,
                  SRGZMODEL,OFFMODEL,FAILMODEL),
             REAL(POWER,CAPACITY,BRKVOLT,PROTECT,BRK1,BRK2,SHUNTZ,
                  SRGZK1,SRGZK2,OFFK1,OFFK2,FAILK1,FAILK2))
STRUCTURE,SN=$B.TUBES,
      ATTRIBUTE=(
            ALPHA(NUMBER,VENDOR,FUNCTION,PINS,REFERENCE,DATA,
                  ANODMODEL,GRIDMODEL),
             REAL(POWER,FREQUENCY,GAIN,XCONDUCT,ANODVOLT,ANODI,

                  ANODCAP,ANODK1,ANODK2,GRIDVOLT,GRIDI,GRIDCAP,GRIDKI,
                  GRIDK2))
CYCLE
$
$     CYCLE 0 NOW CONTAINS THE STRUCTURE FOR A DATA BASE SIMILAR TO
$     SUPERSAP.  THUS TO GENERATE ANOTHER DATA BASE OF THE SAME STRUCTURE
$     THE TYPE 1 FUNCTION COPY CAN BE USED TO PULL CYCLE 0 FROM
$     DBN = SUPER (AS LONG AS CYCLE ZERO EXISTS).
$     THE DATA BASE ENTRIES CAN NOW BE ENTERED AND ATTRIBUTE VALUES
$     ASSIGNED.  THIS CAN BE DONE BY A DBMP FUNCTION FROM PREDETERMINED
$     FILES, FROM CARDS, OR FROM ANOTHER PROGRAM WHICH DETERMINES THE
$     VALUES.
ENDDATA
**
```

APPENDIX D

A SCHEMATIC DIAGRAM OF DBMS NODE STRUCTURES




A
DBE
DBE
to page D-1 (A)
5
#Cells
Cyc
@Cyc Link
@B.STRUCT
#Bits
IPBs
@P.DBE
@S.DBE
#LLCHAIN
@LLCHAIN
@F.SUB
@L.SUB
@P.SUB1
@S.SUB1
ATTRIBUTE VALUES
B
DBE
DBE
to page D-1 (A)
to predecessor in set owned by DBE 'SYS'
to successor in set owned by DBE 'SYS'
8
#Cells
Cyc
@Cyc
link
RTN
LLC
(Similar to one shown below)
7
#Cells
Cyc
@Cyc link
number of indices
Length 1
4
#Cells
Cyc
@Cyc Link
Value 1
DBE
6
#Cells
Cyc
@Cyc Link
@B.DBE
#Bits
IPB
#LLC1
#LLC2
#LLC3
#LLC4
@LLC1
@LLC2
@LLC3
@LLC4
#LLC5
#LLC6
#LLC7
#LLC8
@LLC5
@LLC6
@LLC7
@LLC8
FPTR
C

APPENDIX E

THE DATA BASE MACRO PROCESSOR

This appendix discusses the Data Base Macro Processor (DBMP) concept. The existing DBMS software does not include the DBMP features which are presented here. The following discussion presents the features of the DBMP which would provide the DBMS user with a more powerful, higher level command capability for using the DBMS.

1. THE MACRO CONCEPT

Frequently a sequence of DBMS commands are used repeatedly. To simplify the use of the DBMS, these commands may be grouped together. This group of commands is given a unique name and called a macro. To further simplify the use of DBMS, arguments may be passed to the macro as is done in a FORTRAN subroutine. Macros are also similar to FORTRAN subroutines in that one macro may call another. This is where the similarity between FORTRAN subroutines and data base macros ends. Whereas FORTRAN subroutines reside in core and the commands are executed there directly, macros reside in a data base and only a copy of the macro is made available to the DBMP for executing the contained commands. Although FORTRAN does allow subroutines to call other subroutines thus creating a sequence of called subroutines, any given subroutine may be in the sequence only once. This process of 'nesting' subroutines is called recursion. Since copies of macros are used by the DBMP, the macros may be used in a recursive fashion.

One final distinction between macros and FORTRAN subroutines is that the arguments passed to a macro must be numeric constants, alpha constants, node names, or attribute names.

2. THE LANGUAGE PRIMITIVES

The 12 DBMP commands are listed below followed by a definition of each command. Note that the underlined items are to be supplied by the user.

a. IDENT <u>name</u>

This command allocates (or retrieves) a section of storage in a data base. This storage space is used by the DBMP to save local origins (see functions SNORIGIN, DBEORIGIN, and ORIGIN below), and macros defined by this user. This storage space remains the property of a specific user in that each user has his own storage space. The name field must be from 1 to 9 characters starting with a letter and is used to identify the current user. The name DBMP is reserved for the storage of macros which are shared by all users.

b. MACRO name (number)

The MACRO command is used to identify the beginning of a macro definition. The number field contains an integer representing the number of arguments in the macro call. The number field (and parentheses) are optional. The name field specifies the name used to reference this macro.

c. MEND

The MEND command indicates the end of a macro definition.

d. IF (logical) THEN (text)

This conditional execution command may only appear within a macro definition. When this command is encountered, the next commands are executed only if the logical expression is true. The logical expression is an ANSI FORTRAN logical expression between attribute values and constants. The text field is composed of DBMS and DBMP commands except IDENT, MACRO and MEND.

e. IF (logical) THEN (text1) ELSE (text2)

This conditional execution command may appear only within a macro defintion. When this command is encountered, the text1 commands are executed if the logical expression is true, otherwise the text2 commands are executed. The form of the logical expression and the two text fields is identical to the previous command.

f. WHILE (logical) DO (text)

The WHILE command performs a sequence of commands as long as the logical expression is true. The format of the expression and text fields is the same as the corresponding fields in the conditional execution commands.

g. SNORIGIN name

This command saves the name of the current structure node. The name field may be any alphanumeric string and is the local name of the current structure node. All future references to structure nodes may be referenced to the current structure node through the use of the command ORIGIN (see below).

h. DBEORIGIN name

This command serves the same purpose for data base entries as the command SNORIGIN does for structure nodes.

i. ORIGIN name

The ORIGIN command is used to establish a previously defined structure node (data base entry) as the starting point for future structure node (data base entry) references instead of the structure node SYS. The origin specified by the name field must have been defined by a SNORIGIN or DBEORIGIN command under the same identifier as the current user.

j. CALL name (args)

The CALL command invokes a previously defined macro. The macro specified by the name field must have been defined under the same identifier as the current user. The argument field (args) can be numeric constants, alpha constants, node names, or attribute names. The argument field is optional.

k. DUP ident (origin), name1, name2

The DUP command is used to duplicate another user's structure node data base entries. The other user's identity is given by the ident field and the origin to be used to specify the data base entry is given by the origin field. Using the origin as a starting point the data base entry specified by name1 is copied as the current user's data base entry specified by name2. The origin used for name2 is the origin which was current just prior to the DUP command. The DUP command will not copy linked lists.

l. EXTERNAL (args)

The EXTERNAL command allows the serious systems programmer to extend the capabilities of DBMP by allowing macros to readily call FORTRAN subroutines. The user must supply a subroutine called EXTERN. When DBMP executes this command, subroutine EXTERN is called and the argument list passed to it.

In addition to the above DBMP special commands, any of the DBMS commands may be used. A cell called $0 always contains the status of the last DBMS command executed. The only restriction on the use of DBMS commands is that all references to a data base name within a macro must be the same as the currently used data base (i.e., the name of the data base being used cannot be changed within a macro).

3. THE DBMP PROGRAM

The general form of a DBMP program consists of three sections. The first section identifies the user and contains a DBMS RETRIEVE or INITIAL function followed by a single IDENT command. The second section defines the user macros (unless previously defined in the data base). The third section of a DBMP program contains DBMP and DBMS commands. The only DBMP commands allowed in the third section of a DBMP program are: SNORIGIN, DBEORIGIN, ORIGIN, CALL, and DUP.

Although the three sections of a DBMP program must be in order, not all the sections are required to be present in every DBMP program. The first section is required only if the second section is present or if any DBMP commands appear in the third section. The method of forming the three sections into a DBMP program can be seen more clearly from the following BNF description of the grammar for a DBMP program:

```
prog := id macros main
prog := id main
prog := id macros
prog := main
id := retorini IDENT name
main := texts
macros := macdef
macros := macros macdef
macdef := MACRO name ( integer ) mactext MEND
mactext := text
mactext := IF ( logicalexpression ) THEN ( texts )
mactext := IF (logicalexpression) THEN ( texts ) ELSE ( texts )
mactext := WHILE ( logicalexpression ) DO ( texts )
mactext := EXTERNAL ( integer )
mactext := EXTERNAL (integer , args )
texts := text
texts := texts text
text := CALL name
text := CALL name (args )
text := SNORIGIN name
text := DBEORIGIN name
text := ORIGIN name
text := FILE name
text := dbmscommand
args := arg
args := args , arg
args := args = arg
arg := number
arg := alpha
arg := name
```

In the above BNF description

1) the fields 'integer', 'number', and 'alpha' are defined in Appendix A in the discussion of the Type 3 DBMS function CREATE,

2) the field 'name' consists of 1 to 9 alphameric characters beginning with a letter,

3) the field 'logicalexpression' is any DBMS relational expression,

4) the field 'dbms command' is any DBMS command,

and 5) the field 'retorini' represents the DBMS command RETRIEVE or INITIAL.

Macros are defined in the second section of a DBMP program. The sequence of commands to be placed in the macro are proceeded by a MACRO command and followed by a MEND command. The MACRO command contains the macro's name and the number of arguments to be passed to the macro.

The sequence of commands in the macro is called the macro text (or simply text). Within the text a passed parameter is referenced as $K where K is an integer specifying which parameter is being referenced. The text of a macro may contain any DBMP or DBMS command except IDENT, MACRO, and MEND, and the name of the data base being used by DBMS may not be changed.

The following example will help to clarify the use of macros. It does assume a knowledge of the data base construction and as such should not be considered too strongly until the data base structure is understood. The example is presented here for completeness. In particular consider the data base created in Example 1 of Appendix C. Assume that a person named Joe has established an origin called TRAN at SN 'SYS.SIMS.XTORS.MODEL*$F'. The following macro will find and print all transistor models for a given person (in this case Bob Smith):

```
RETRIEVE, DBN = FIG6                          first section
IDENT JOE

MACRO NAMES (1)
   ORIGIN - TRAN
   WHILE ($0.EQ. 0) DO                        second section
      IF (NAME.EQ.$1)
   THEN (DISPLAY)
   FIND,DBE = $S

MEND

CAL NAMES (/BOB SMITH/)                       third section
```

Note the use of the DBMP flag $0 to test when all the DBEs have been examined (a nonexistent DBE is requested).

If at a later time Joe wishes to display all of Jim Brown's transistor models, the following deck would be used:

```
Retrieve, DBN = FIG6                          first section
IDENT JOE

                                              no second section

CALL NAMES (*JIM BROWN*)                      third section
```

It is now apparent that when a new data base is being used and several generally useful macros have been defined in another data base, the DBMS may be used to copy the macros from the old data base to the new data base. The origins copied in this fashion are no longer valid and should be updated or discarded.

4. MACRO STORAGE AND EXECUTION

The macros are stored in DBMS short form as data base entries in the current data base. The data base entries containing the macros are associated with the structure node SYS.DBMP.ident.MACRO, where ident is the current user's identity. Similarly the user's origins are saved in data base entries associated with structure node SYS.DBMP.ident.ORIGINS, where ident is the current users identity. Thus when DBMP is being used, these two structure node names are reserved.

When a macro is called, a copy of the macro is saved. Included in the copy is a return pointer to the calling macro or main program. Also, all passed arguments are substituted for dummy arguments in the macro. When a macro terminates, the copy of the macro is destroyed and DBMP resumes executing the calling macro or main program.

As a macro is being executed, DBMP passes DBMS commands directly to DBMS. When IF THEN and WHILE DO commands are encountered, DBMS isolates the expression and then evaluates it by retrieving the attribute values from the data base with DBMS commands. When a condition is failed, DBMP proceeds to the next command to be executed.

APPENDIX F

GLOSSARY

Alpha - An alpha numeric string (also called a constant). To encode an alpha text into an alpha string, choose any delimiter from the set ( ) + * / , - or =. The delimiter is placed immediately before and after the alpha text string. Anywhere in the text the delimiter appears it must be replaced by two delimiters.

Ancestor - When referring to a node on a tree, the ancestors of the node are the parent of the node plus the parent of the parent node (also known as the grandparent), etc. The meaning of ancestor becomes clear when one considers a family tree (a lineal chart).

ANSI FORTRAN - The standard FORTRAN language as approved March 7, 1966 by the United States of America Standards Institute.

Array - An array (or a simple array) is any FORTRAN array. See Structure Level 1.4 for an explicit definition.

Array Description Cell - A node attached to a structure node or a data base entry containing the dimensionality of an array.

Array Value Cell - A node attached to a data base entry containing the values of an array.

Attribute - An attribute is composed of a name part and a value part. One or more attributes define the state of an entity.

Attribute Name - The name part of an attribute. See also Attribute.

Attribute Value - The value part of an attribute. See also Attribute.

Bit String - A series of bits with values 0 and 1, or the entire bit string considered as having one or more substrings (also called subfields) which are interpreted as integers, flags, etc.

BNF - Abbreviation for Backus-Naur Form, a concise notation for describing the manner of constructing the allowable sentences of a language.

Branch - Refers to a non-leaf node in a tree (in this document branch strictly refers to a ragged table subtree). See also Tree.

Branchlets - When considering a subtree of a tree the branchlets are the immediate subtrees of the subtree being considered. When considering a subtree with node A, the branchlets then are the subtrees of A (may be a branch or a leaf).

Cell - A distinct non-disjoint, recognizable grouping of information. See Structure Level 2 for a complete definition of cell and the formats of cells used in DBMS.

Cell Type - Refers to the format and type of information contained in a cell. See also Cell.

Chain - See Chain of Nodes.

Chain of Nodes - In a data base a group of similar nodes are frequently connected by pointers. The nodes and pointers form a chain. In particular, a chain has a starting and ending node and all the nodes in a chain can be visited by following a specific pointer in each of the nodes. A chain is owned by a node which has a pointer to the starting node of the chain; the ending node of a chain normally points to the chain owner. For example a SN owns a chain of DBEs which are called the SN's associated DBEs.

Children - Those nodes of a tree which are immediate descendents of a particular node in the tree are called the children of that particular node. See also Descendent below.

Class Descriptor - Refers to the format and information contained in cells used to store any of the seven classes of keyword packets (see Structure Level 2.3).

Cycle - The information in a data base is stored in nodes. The nodes may be archived by cycle. A cycle then represents the state of a data base at a user defined instant of time. The date and time a cycle was created plus the information saved in any cycle can be retrieved with DBMS commands (see Appendix A Type 1 functions LIST and COPY).

Current Data Base - The name of the data base which DBMS (or DBMP) is currently working with. See Appendix A Type 1 functions for a definition of current data base.

Current SN - The structure node DBMS (or DBMP) is referencing (or most recently referenced). See the general discussion of Type 2 function in Appendix A for a complete definition and default values.

Current DBE - The data base entry DBMS (or DBMP) is referencing (or most recently referenced). See the general discussion of Type 3 functions in Appendix A for a complete definition and default values.

Data Base Entry - The nodes in a data base which contain the attribute values (user information). Structure Level 1.2 defines a data base entry (DBE) and 2.5 presents the format of a DBE.

DBE - See Data Base Entry.

DBE set - A grouping of DBEs. The set membership is defined by entering individual DBEs into the set. Structure Level 1.2 defines DBE set (also called set).

DBMP - Data Base Macro Processor, a user interface with the DBMS.

DBMS - Data Base Management System, a program for managing the storage and retrieval of information.

Descendents - When referring to a node of a tree, the descendents of the node are all the children of the node plus all the children's children, etc. The meaning of descendents becomes clear when one considers a family tree (a lineal chart).

Defined Continuation - When DBMS constructs a ragged table, special nodes are attached as each branch is defined. These special nodes are called undefined continuation nodes and correspond to a branch or leaf which has not been defined. If and when undefined continuation is defined (by the user) to be a branch, the new branch is generated as a disjoint continuation of the tree. The undefined continuation is converted to a defined continuation. Structure Level 3.5 gives a complete description of this process.

Delimiter - Any of the symbols +-,/*()=. or $ (See Appendix A, see also Alpha)

FIFO - First in, first out; refers to the order of retrieving information from a queue. In general, a method of storing objects (entities) where the oldest object is always removed first.

Generic Name - A method of forming a SN's name where the names of all the ancestors and the node are separated by periods. See Structure Level 1.2 for examples of generic names.

Header - Refers to the beginning cells of a node which identify what information is contained in the node and the format of the information. Structure Level 2 provides a definition of the term header along with several examples.

Internal Form - Refers to the format into which DBMS translates commands before using them, which is the same format used for interprogram communication between DBMP and DBMS. Structure Level 2.2 defines the internal form for all commands to DBMS.

IPB - Item present bits, a group of bits included in most data base nodes to indicate which items are present in the node. The primary function of IPBs is to allow shorter nodes by not including pointers which will not be used frequently and also to preserve data base sanity. Structure Levels 2.3 through 2.6 present examples of IPBs.

Keyword - A word used to identify a field of a DBMS command while in the long mode. Appendix A gives a list of the usage of the DBMS commands which includes all keywords.

Keyword Class - See Class Descriptor.

Keyword Description - See Class Descriptor.

Keyword Packet - A cell used to store a keyword value in DBMS SHORT form (see also Internal Form).

Keyword Phrase - A keyword and the value of the Keyword separated by an equal sign (=).

Leaf - A terminal node on a tree. When referencing a ragged table leaf refers to an attribute type and an attribute value (or values).

Level - Refers to SNs or DBEs. The level of a structure node is the number of its ancestors. The level of a DBE is the level of its associated SN.

LIFO - Last in, first out; refers to the order of retrieving information from a queue. In general, a method of storing objects (entities) where the youngest object is always removed first.

Linked List - An inverted linked list of DBEs. Membership is determined by attribute value (also relative position in the data base tree). Structure Level 1.5 gives a complete definition of Linked Lists (LL).

Linked List Chain Node - A node attached to a DBE used to chain the DBE into linked lists (also referred to as LLC). Structure Level 2.6 gives the format for a LLC.

Linked List Header Node - A node attached to a SN which contains the membership requirements and the ordering algorithm (also called a LLH). Structure Level 2.4 gives the format for a LLH. See also Rank Ordering.

List - See Linked List.

LL - See Linked List.

LLC - See Linked List Chain Node.

LLH - See Linked List Header Node.

Long - See Long Mode.

Long Mode - The alpha numeric form of DBMS commands as presented in Appendix A.

Mode - See Long Mode and Internal Form.

Motion Phrase - A phrase included in the name of a node. The effect of a motion phrase is to move from one node to another related node. Structure Level 1.3 defines the term motion phrase and specifies the results of various motion phrases. Appendix B gives a tabular form of all the motion phrases.

Node - The primary building blocks in a data base. Structure Level 1.2 introduces two primary nodes, and Structure Levels 2.3 through 2.6 present the formats of the primary nodes. Structure level 3.3 presents the format of an own node which is used during data base construction.

Parent - If one or more nodes are immediate descendents of a particular node in a tree, that particular node is called the parent of the descendent nodes.

Permission Parameter - A set of three keys stored in the data base. These keys are intended for protection against accidental destruction of a data base or part of a data base.

Pointer - An integer specifying the location of a node within the data base file.

Predecessor - considering a node in a chain of nodes (linked by pointers), the node containing a pointer to the considered node is the predecessor in the chain.

Ragged Table - A method of storing data similar to arrays but differing from arrays in that the number of entries in each row is not necessarily the same. A ragged table is best viewed as a tree where the leaves are the entries. Structure Level 1.4 defines a ragged table and Structure Level 3.5 discusses the method of creating and storing ragged tables.

Rank Ordering - Linked lists and sets are both ordered. The specification of the ordering is called the rank ordering of the list or set.

Root - The specific node of a tree which does not have a parent, considered the origin or base of a tree.

Set - See DBE set.

Siblings - The children of a node in a tree are called siblings.

SN - See Structure Node.

Structure Node - One of the two primary nodes in a data base. The structure nodes contain the hierarchical structure of a data base and act as a template to interpret the information kept in a DBE. Structure Level 1.2 defines a structure node (SN) and 2.3 presents the format of a SN.

Structure Set - A structure set is composed of a subset of the nodes on a branch of the structure node tree. A structure tree is said to belong to a structure node. The rules for forming structure sets are presented in Structure Level 1.2.

Structure Tree - A tree composed entirely of structure nodes which represents the hierarchy of a data base. Structure trees are defined in Structure Level 1.2.

Successor - Considering a node in a chain of nodes (linked by pointers), the node pointed at by the considered node is the successor in the chain.

SYS - The name of the SN which is the root of the structure tree. It is also the name of the DBE associated with the SN called SYS. The attributes of the DBE called SYS contain pertinent DBMS information such as cycle information. Both of these nodes are automatically created by DBMS.

Tag - See Header.

Tagged - Refers to any data structure whose modules of information are preceded by tags.

Text - In a tagged architecture, the text of a node is all information stored in the node exclusive of the header (see Structure Level 2).

Tree - A tree is a grouping of nodes such that:

1) There is one node which is called the root of the tree.

2) The remaining nodes form disjoint groups where each of the groups is a tree. Each of the groups is referred to as a subtree of the original tree.

Undefined Continuation - See Defined Continuation.

DISTRIBUTION LIST

DEPARTMENT OF DEFENSE

Director
Armed Forces Radiobiology Research Institute

Defense Nuclear Agency
ATTN: Robert E. Carter
ATTN: Tech. Lib.

Director of Defense Research & Engineering
ATTN: Asst. Dir., Strat. Wpns.

Director
Defense Communications Agency
ATTN: Code 540.5
ATTN: Code 930, Monte I. Burgett, Jr.

Defense Documentation Center
12 cy ATTN: TC

Director
Defense Intelligence Agency
ATTN: DI-7D
ATTN: DI-3

Director
Defense Advanced Research Projects Agency
ATTN: NMR

Director
Defense Nuclear Agency
ATTN: DDST
ATTN: STVL
ATTN: STSI
ATTN: RAEV, Captain Van Prouyen
ATTN: SPAS, J. Moulton
3 cy ATTN: STTL, Tech. Lib.
5 cy ATTN: SPAS, D. Kohler
5 cy ATTN: RAEV, J. Farber

Headquarters
European Command
ATTN: ECJ6-PF

Commander
Field Command
Defense Nuclear Agency
ATTN: FCPR

Chief
Defense Nuclear Agency, FC
Las Vegas Liason Office
ATTN: FCTCL

Director
Joint Strategic Target Planning Staff, JCS
ATTN: JLTW

Chief
Livermore Division, Field Command, DNA
Lawrence Livermore Laboratory
ATTN: FCPRL

OJCS/J-3
ATTN: J-3, RDTA Br., WWMCCS, Plans Div.

DEPARTMENT OF THE ARMY

Commander
Frankford Arsenal
ATTN: SARFA-FCD, Marvin Elnick

Commander
Harry Diamond Laboratories
ATTN: AMXDO-EM, R. Bostak
ATTN: AMXDO-RB, Joseph R. Miletta
ATTN: AMXDO-RBI, John A. Rosado
ATTN: AMXDO-RCC, John E. Thompkins
ATTN: AMXDO-EM, Robert F. Gray
ATTN: AMXDO-EM, Raphael Wong
ATTN: AMXDO-RC, Robert B. Oswald, Jr.
ATTN: AMXDO-EM, J. W. Beilfuss
ATTN: *HDL, Library*
ATTN: D. Schallhorn
ATTN: P. Caldwell
ATTN: S. Graybill
ATTN: J. Gwaltney

Commander
Picatinny Arsenal
ATTN: SMUPA-FR-S-P, Lester W. Doremus
ATTN: SARPA-ND-N
ATTN: SARPA-ND-C-E, Amina Nordio
ATTN: Dr. P. Harris

Commander
TRASANA
ATTN: ATAA-EAC, Franics N. Winans

Director
U.S. Army Ballistic Research Labs.
ATTN: AMXBR-X, Julius J. Meszaros
ATTN: AMXBR-VL, John W. Kinch
ATTN: AMXRD-BVL, David L. Rigotti
ATTN: AMXBR-ED, H. Burden

Commander
U.S. Army Electronics Command
ATTN: AMSEL-GG-TD, W. R. Werk
ATTN: AMSEL-TL-MD, Gerhart K. Gaule
ATTN: AMSEL-TL-IR, Edwin T. Hunter

Commanding Officer
U.S. Army Electronics Command
Night Vision Laboratory
ATTN: CPT Allan S. Parker

Commander
U.S. Army Electronics Proving Ground
ATTN: STEEP-MT-M, Gerald W. Durbin

Commandant
U.S. Army Field Artillery School
ATTN: ATSFA-CTD-ME, Harley Moberg

Commander
U.S. Army Mat. & Mechanics Rsch. Ctr.
ATTN: AMXMR-HH, John F. Dignam
ATTN: Dr. T. Chow

Commander
U.S. Army Materiel Dev. & Readiness Cmd.
ATTN: AMCRD-WN-RE, John F. Corrigan

DEPARTMENT OF DEFENSE CONTRACTORS (Continued)

The Bendix Corporation
Research Laboratories Division
ATTN: Donald J. Niehaus, Mgr., Prgm. Dev.

The Boeing Company
ATTN: Robert S. Caldwell, 2R-00
ATTN: Donald W. Egelkrout, 2R-00
ATTN: David L. Dye, 87-75
ATTN: Aerospace Library
ATTN: Howard W. Wicklein, 17-11
ATTN: Dr. Lempriere
ATTN: J. Adamski
ATTN: J. Shrader

Booz-Allen & Hamilton, Inc.
ATTN: Raymond J. Chrisner

Brown Engineering Company, Inc.
ATTN: David L. Lambert, M.S. 18

Charles Stark Draper Laboratory, Inc.
ATTN: Kenneth Fertig
ATTN: Paul R. Kelly

Computer Sciences Corporation
ATTN: Richard H. Dickhaut

Cutler-Hammer, Inc.
AIL Division
ATTN: Anne Anthony, Central Tech. Files

University of Denver
Colorado Seminary
ATTN: Sec. Officer for Fred P. Venditti

The Dikewood Corporation
ATTN: L. Wayne Davis

Effects Technology, Inc.
ATTN: Mr. B. Wengler
ATTN: Mr. M. Rosen

E-Systems, Inc.
Greenville Division
ATTN: Library, 8-50100

Exp. & Math. Physics Consultants
ATTN: Thomas M. Jordan

Fairchild Industries, Inc.
ATTN: Mgr., Config. Data & Standards

The Franklin Institute
ATTN: Ramie H. Thompson

Garrett Corporation
ATTN: Robert E. Weir, Dept. 93-9

General Electric Company
Space Division
ATTN: Larry I. Chasen
ATTN: G. Harrison
ATTN: John R. Greenbaum
ATTN: Joseph C. Peden, CCF 8301
ATTN: John L. Andrews
ATTN: James P. Spratt
ATTN: J. Hannabeck
ATTN: R. Peterson

DEPARTMENT OF DEFENSE CONTRACTORS (Continued)

General Electric Company
Re-Entry & Environmental Systems Div.
ATTN: Robert V. Benedict

General Electric Company
TEMPO-Center for Advanced Studies
ATTN: DASIAC
ATTN: William McNamera
ATTN: Royden R. Rutherford
ATTN: M. Espig

General Electric Company
ATTN: CSP 0-7, L. H. Dee

General Electric Company
Aerospace Electronics Systems
ATTN: W. J. Patterson, Drop 233
ATTN: George Francis, Drop 233

General Electric Company-TEMPO
ATTN: DASIAC for William Alfonte

General Research Corporation
ATTN: Robert D. Hill

General Research Corporation
Washington Operations
ATTN: David K. Osias

GTE Sylvania, Inc.
Electronics Systems Grp.-Eastern Div.
ATTN: Leonard L. Blaisdell
ATTN: James A. Waldon

GTE Sylvania, Inc.
ATTN: Charles H. Ramsbottom
ATTN: Herbert A. Ullman
ATTN: David P. Flood

Gulton Industries, Inc.
Engineered Magnetics Division
ATTN: Eng. Magnetics Div.

Harris Corporation
Harris Semiconductor Division
ATTN: T. L. Clark, M.S. 4040
ATTN: Wayne E. Abare, M.S. 16-111
ATTN: Carf F. Davis, M.S. 17-220

Hazeltine Corporation
ATTN: M. Waite, Tech. Info. Ctr.

Honeywell, Incorporated
Government & Aeronautical Products Division
ATTN: Ronald R. Johnson, A-1622

Honeywell, Incorporated
Aerospace Division
ATTN: Stacey H. Graff, M.S. 725-J
ATTN: James D. Allen, M.S. 775-D
ATTN: Harrison H. Noble, M.S. 725-5A

Honeywell, Incorporated
ATTN: Tech. Lib.

Hughes Aircraft Company
ATTN: Billy W. Campbell, M.S. 6-E-110
ATTN: Kenneth R. Walker, M.S. D-157

DEPARTMENT OF THE ARMY (Continued)

Commander
U.S. Army Missile Command
ATTN: AMSI-RGP, Victor W. Ruwe
ATTN: AMSMI-RGP, Hugh Green
ATTN: AMSMI-RRR, Faison P. Gibson

Commander
U.S. Army Nuclear Agency
ATTN: ATCN-W, LTC Leonard A. Sluga

Project Manager
U.S. Army Tactical Data Systems, AMC
ATTN: Dwaine B. Huewe

Commander
White Sands Missile Range
ATTN: STEWS-TE-NT, Marvin P. Squires

DEPARTMENT OF THE NAVY

Chief of Naval Research
Navy Department
ATTN: Code 427

Commanding Officer
Naval Ammunition Depot
ATTN: Code 7024, James Ramsey

Commander
Naval Electronic Systems Command
ATTN: ELEX 05323, Cleveland F. Watkins
ATTN: PME 117-21
ATTN: Code 50451

Commander
Naval Electronics Laboratory Center
ATTN: H. F. Wong
ATTN: Code 4223

Director
Naval Research Laboratory
ATTN: Code 2627, Doris R. Folen
2 cy ATTN: Dr. G. Cooperstein

Officer-in-Charge
Civil Engineering Lab.

Commander
Naval Surface Weapons Center
ATTN: Code 431, Edwin B. Dean
ATTN: Code WX-21, Tech. Lib.
ATTN: L. Gawan
ATTN: Dr. Pastine
ATTN: Dr. Sazama (CASINO)
ATTN: Code 730

Commander
Naval Surface Weapons Center
ATTN: Code FUR, Robert A. Amadori

Commanding Officer
Naval Weapons Evaluation Facility
ATTN: Code ATG, Mr. Stanley
ATTN: Code ADS

Director
Strategic Systems Project Office
ATTN: NSP-2701, John W. Pitsenberger

DEPARTMENT OF THE AIR FORCE

Commander
Aeronautical Systems Division, AFSC
ATTN: ASD-YH-EX, Lt Col Robert Leverette
ATTN: Tech. Lib.

Commander
ADC/XP
ATTN: XPQY

AF Civil Engineering Center
ATTN: PREC

Commandant
AF Flight Dynamics Laboratory, AFSC
ATTN: DOO/Lib.

Director
Air University Library
ATTN: LDE

Commander
Air University
ATTN: ED, Dir., Civ. Eng.

AF Institute of Technology, AU
ATTN: Tech. Lib., Bldg. 640, Area B
ATTN: CES

AF Materials Laboratory, AFSC
ATTN: Tech. Lib.

AFTAC
ATTN: TAE

Air Force Avionics Laboratory, AFSC
ATTN: AFAL, TEA, Hans J. Hennecke
ATTN: AFAL, AAA

Commander
Rome Air Development Center, AFSC
ATTN: Doc. Lib.

AF Rocket Propulsion Laboratory
ATTN: DYSN

HQ USAF/PR
ATTN: PREE

HQ USAF/XO
ATTN: XOOWD

Deputy Inspector General
Inspection & Safety
ATTN: PQAL

Headquarters
Air Force Systems Command
ATTN: DLSP
ATTN: DLCAW

U.S. Air Force Academy
ATTN: DFSLB
ATTN: DFCE

AF Armament Laboratory, AFSC
ATTN: DLOSL
ATTN: DEE

DEPARTMENT OF THE AIR FORCE (Continued)

AF Aero-Propulsion Laboratory, AFSC
ATTN: POD, P. E. Stover

Commander
Foreign Technology Division, AFSC
ATTN: FTD/PDJC

AF Weapons Laboratory, AFSC
ATTN: ELA
ATTN: ELC
ATTN: ELP, Carl E. Baum
ATTN: HO
10 cy ATTN: Mr. K. D. Smith, DYV
ATTN: AL
ATTN: ALC
ATTN: ALE
ATTN: ALO
ATTN: AR
ATTN: DE
ATTN: DEV
ATTN: DEX
ATTN: DY
ATTN: DYT
2 cy ATTN: DYS, Dr. Baker & Dr. Burns
3 cy ATTN: DYV, Maj Mitchell, Maj Stuber & F. Bick
ATTN: DYX
ATTN: EL
10 cy ATTN: ELP
ATTN: ELS, B. Kline
ATTN: LR
ATTN: LRE
ATTN: LRL
ATTN: LRO
ATTN: LRP
ATTN: PG
ATTN: PGA
ATTN: PGV
ATTN: PO
ATTN: SA
ATTN: SAB
ATTN: SAS
2 cy ATTN: SUL

Commander
Ogden Air Logistics Center
ATTN: MMEWM, Robert Joffs

Commander in Chief
Pacific Air Forces
ATTN: DEE

Commander in Chief
Strategic Air Command
ATTN: XPFS, Maj Brian G. Stephan

USAF, SCLO
ATTN: Maj Pierson, Chief, LO

SAMSO/DY
ATTN: DYS, Maj Larry A. Darda

SAMSO/MN
ATTN: MNNR

SAMSO/RS
ATTN: RSSE

DEPARTMENT OF THE AIR FORCE (Continued)

SAMSO/YD
ATTN: YDD, Maj Marion F. Schneider

ENERGY RESEARCH & DEVELOPMENT ADMINISTRATION

University of California
Lawrence Livermore Laboratory
ATTN: Donald J. Meeker, L-153
ATTN: E. K. Miller, L-156
ATTN: Lawrence Cleland, L-156
ATTN: Frederick R. Kovar, L-94
ATTN: William J. Hogan, L-531
ATTN: Dr. Keller
ATTN: W. Isbell
ATTN: Dr. Mayer

Los Alamos Scientific Laboratory
ATTN: Doc. Con. for Bruce W. Noel
ATTN: Doc. Con. for J. Arthur Freed
ATTN: Report Library
ATTN: Dr. Skaggs
ATTN: Dr. Dingus

Sandia Laboratories
ATTN: 3141
ATTN: Dr. Posey
ATTN: Dr. Butcher
ATTN: Dr. Toepfer

DEPARTMENT OF DEFENSE CONTRACTORS

Aerojet Electro-Systems Co., Div.
Aerojet-General Corporation
ATTN: Thomas D. Hanscome

Aeronutronic Ford Corporation
Aerospace & Communications Ops.
ATTN: E. R. Poncelet, Jr.
ATTN: Ken C. Attinger
ATTN: Tech. Info. Section

Aeronutronic Ford Corporation
Western Development Laboratories Div.
ATTN: Samuel R. Crawford, M.S. 531

Aerospace Corporation
ATTN: Dr. B. Barry
ATTN: Dr. M. Kausch
ATTN: Dr. J. Benveniste

Avco Research & Systems Group
ATTN: Research Library, A-830, Rm. 7201
ATTN: Dr. Bade
ATTN: W. Broding

The BDM Corporation
ATTN: T. H. Neighbors
ATTN: William Druen

The BDM Corporation
ATTN: James M. Phelan

Bell Aerospace Company
Division of Textron, Inc.
ATTN: Carl B. Schoch, Wpns. Effects Grp.

The Bendix Corporation
Communication Division
ATTN: Doc. Con.

DEPARTMENT OF DEFENSE CONTRACTORS (Continued)

The Bendix Corporation
Research Laboratories Division
ATTN: Donald J. Niehaus, Mgr., Prgm. Dev.

The Boeing Company
ATTN: Robert S. Caldwell, 2R-00
ATTN: Donald W. Egelkrout, 2R-00
ATTN: David L. Dye, 87-75
ATTN: Aerospace Library
ATTN: Howard W. Wicklein, 17-11
ATTN: Dr. Lempriere
ATTN: J. Adamski
ATTN: J. Shrader

Booz-Allen & Hamilton, Inc.
ATTN: Raymond J. Chrisner

Brown Engineering Company, Inc.
ATTN: David L. Lambert, M.S. 18

Charles Stark Draper Laboratory, Inc.
ATTN: Kenneth Fertig
ATTN: Paul R. Kelly

Computer Sciences Corporation
ATTN: Richard H. Dickhaut

Cutler-Hammer, Inc.
AIL Division
ATTN: Anne Anthony, Central Tech. Files

University of Denver
Colorado Seminary
ATTN: Sec. Officer for Fred P. Venditti

The Dikewood Corporation
ATTN: L. Wayne Davis

Effects Technology, Inc.
ATTN: Mr. B. Wengler
ATTN: Mr. M. Rosen

E-Systems, Inc.
Greenville Division
ATTN: Library, 8-50100

Exp. & Math. Physics Consultants
ATTN: Thomas M. Jordan

Fairchild Industries, Inc.
ATTN: Mgr., Config. Data & Standards

The Franklin Institute
ATTN: Ramie H. Thompson

Garrett Corporation
ATTN: Robert E. Weir, Dept. 93-9

General Electric Company
Space Division
ATTN: Larry I. Chasen
ATTN: G. Harrison
ATTN: John R. Greenbaum
ATTN: Joseph C. Peden, CCF 8301
ATTN: John L. Andrews
ATTN: James P. Spratt
ATTN: J. Hannabeck
ATTN: R. Peterson

DEPARTMENT OF DEFENSE CONTRACTORS (Continued)

General Electric Company
Re-Entry & Environmental Systems Div.
ATTN: Robert V. Benedict

General Electric Company
TEMPO-Center for Advanced Studies
ATTN: DASIAC
ATTN: William McNamera
ATTN: Royden R. Rutherford
ATTN: M. Espig

General Electric Company
ATTN: CSP 0-7, L. H. Dee

General Electric Company
Aerospace Electronics Systems
ATTN: W. J. Patterson, Drop 233
ATTN: George Francis, Drop 233

General Electric Company-TEMPO
ATTN: DASIAC for William Alfonte

General Research Corporation
ATTN: Robert D. Hill

General Research Corporation
Washington Operations
ATTN: David K. Osias

GTE Sylvania, Inc.
Electronics Systems Grp.-Eastern Div.
ATTN: Leonard L. Blaisdell
ATTN: James A. Waldon

GTE Sylvania, Inc.
ATTN: Charles H. Ramsbottom
ATTN: Herbert A. Ullman
ATTN: David P. Flood

Gulton Industries, Inc.
Engineered Magnetics Division
ATTN: Eng. Magnetics Div.

Harris Corporation
Harris Semiconductor Division
ATTN: T. L. Clark, M.S. 4040
ATTN: Wayne E. Abare, M.S. 16-111
ATTN: Carf F. Davis, M.S. 17-220

Hazeltine Corporation
ATTN: M. Waite, Tech. Info. Ctr.

Honeywell, Incorporated
Government & Aeronautical Products Division
ATTN: Ronald R. Johnson, A-1622

Honeywell, Incorporated
Aerospace Division
ATTN: Stacey H. Graff, M.S. 725-J
ATTN: James D. Allen, M.S. 775-D
ATTN: Harrison H. Noble, M.S. 725-5A

Honeywell, Incorporated
ATTN: Tech. Lib.

Hughes Aircraft Company
ATTN: Billy W. Campbell, M.S. 6-E-110
ATTN: Kenneth R. Walker, M.S. D-157

DEPARTMENT OF DEFENSE CONTRACTORS (Continued)

Hughes Aircraft Company
Space Systems Division
ATTN: William W. Scott, M.S. A-1080
ATTN: Edward C. Smith, M.S. A-620

IBM Corporation
ATTN: Frank Frankovsky
ATTN: Harry W. Mathers, Dept. M-41

IIT Research Institute
ATTN: Irving N. Mindel

Ion Physics Corporation
ATTN: Mr. B. Evans

IRT
ATTN: R. L. Mertz
ATTN: Leo D. Cotter
ATTN: Eric P. Wenaas
ATTN: MDC

Kaman Sciences Corporation
ATTN: John R. Hoffman
ATTN: Albert P. Bridges
ATTN: Donald H. Bryce
ATTN: W. Foster Rich
ATTN: Walter E. Ware
ATTN: Dr. Shelton
ATTN: T. Meagher

KTech Corporation
ATTN: Dr. D. V. Keller
ATTN: Mr. N. Froula
ATTN: Mr. L. Lee

Litton Systems, Inc.
Guidance & Control Systems Division
ATTN: R. W. Maughmer
ATTN: Val J. Ashby, M.S. 67

Lockheed Missiles & Space Co., Inc.
ATTN: George F. Heath, Dept. 81-14
ATTN: Benjamin T. Kimura, Dept. 81-14
ATTN: Hans L. Schneemann, Dept. 81-64
ATTN: Dr. Miller
ATTN: Dr. Burford
ATTN: R. Smith, 81-14
ATTN: R. Walz
ATTN: T. Kellcher

LTV Aerospace Corporation
ATTN: Technical Data Ctr.

Martin Marietta Aerospace
Orlando Division
ATTN: Jack M. Ashford, MP-537
ATTN: Mona C. Griffith, Library, MP-30
ATTN: William W. Mras, MP-413

Martin Marietta Corporation
Denver Division
ATTN: J. E. Goodwin, Mail 0452
ATTN: Paul G. Kase, Mail 8203

Maxwell Laboratories, Inc.
ATTN: Dr. V. Fargo

McDonnell Douglas Corporation
ATTN: Tom Ender
ATTN: Tech. Lib.

DEPARTMENT OF DEFENSE CONTRACTORS (Continued)

McDonnell Douglas Corporation
ATTN: Stanley Schneider
ATTN: Raymond J. DeBattista
ATTN: Dr. Reck
ATTN: Dr. Berkowitz

McDonnell Douglas Corporation
ATTN: Tech. Lib., C1-290/36-84

Mission Research Corporation
ATTN: William C. Hart

Mission Research Corporation
ATTN: J. Roger Hill
ATTN: David E. Merewether

The Mitre Corporation
ATTN: M. E. Fitzgerald

National Academy of Sciences
ATTN: National Materials Advisory Board for
R. S. Shane, Nat. Materials Advsy.

Northrop Corporation
Electronic Division
ATTN: Boyce T. Ahlport
ATTN: George H. Towner
ATTN: Vincent R. DeMartino

Northrop Corporation
ATTN: Orlie L. Curtis, Jr.
ATTN: James P. Raymond
ATTN: David N. Pocock

Northrop Corporation
Electronic Division
ATTN: Joseph D. Russo

Physics International Company
ATTN: Doc. Con. for John H. Huntington
ATTN: Dr. Shea
ATTN: Dr. Putnam
10 cy ATTN: K. Childers

Prototype Development Associates, Inc.
ATTN: Mr. T. McKinley

R & D Associates
ATTN: S. Clay Rogers
ATTN: Leonard Schlessinger
ATTN: Dr. Rausch
ATTN: Dr. Field

The Rand Corporation
ATTN: Cullen Crain
ATTN: O. Nance

Raytheon Company
ATTN: Gajanan H. Joshi, Radar Sys. Lab.

Raytheon Company
ATTN: James R. Weckback
ATTN: Harold L. Flescher

RCA Corporation
Government & Commercial Systems
ATTN: George J. Brucker

DEPARTMENT OF DEFENSE CONTRACTORS (Continued)

RCA Corporation
ATTN: E. Van Keuren, 13-5-2

Rockwell International Corporation
ATTN: James E. Bell, HA-10
ATTN: George C. Messenger, FB-61

Rockwell International Corporation
Electronics Operations
ATTN: Mildred A. Blair
ATTN: Alan A. Langenfeld
ATTN: Dennis Sutherland

Sanders Associates, Inc.
ATTN: Moe L. Aitel, NCA 1-3236

Science Applications, Inc.
ATTN: Frederick M. Tesche

Science Applications, Inc.
ATTN: Mr. R. Fisher

Science Applications, Inc.
ATTN: Dr. J. Cramer

Science Applications, Inc.
ATTN: William L. Chadsey

Science Applications, Inc.
ATTN: J. Robert Beyster
ATTN: Larry Scott

Science Applications, Inc.
ATTN: Noel R. Byrn

Simulation Physics, Inc.
ATTN: John R. Uglum

Simulation Physics, Inc.
ATTN: Mr. R. Little

The Singer Company
ATTN: Irwin Goldman, Eng. Management

Southern Research Institute
ATTN: Mr. Colt Pears

Sperry Flight Systems Division
Sperry Rand Corporation
ATTN: D. Andrew Schow

Sperry Rand Corporation
Sperry Division
ATTN: Paul Marraffino

DEPARTMENT OF DEFENSE CONTRACTORS (Continued)

Stanford Research Institute
ATTN: Philip J. Dolan
ATTN: Arthur Lee Whitson
ATTN: A. Lutze

Sundstrand Corporation
ATTN: Curtis B. White

Systems, Science & Software
ATTN: Dr. G. Gurtman

Systron-Donner Corporation
ATTN: Gordon B. Dean

Texas Instruments, Inc.
ATTN: Gary F. Hanson
ATTN: Donald J. Manus, M.S. 72

Texas Tech University
ATTN: Travis L. Simpson

TRW Systems Group
ATTN: A. M. Liebschutz, R1-1162
ATTN: Richard H. Kingsland, R1-2154
ATTN: A. A. Witteles, R1-1120
ATTN: Aaron H. Narevsky, R1-2144
ATTN: Lillian D. Singletary, R1-1070
ATTN: Jerry I. Lubell
ATTN: Benjamin Sussholtz

TRW Systems Group
San Bernardino Operations
ATTN: John E. Dahnke
ATTN: H. S. Jensen

United Technologies Corporation
Hamilton Standard Division
ATTN: Raymond G. Giguere

Victor A. J. Van Lint, Consultant
Mission Research Corporation
ATTN: V. A. J. Van Lint

Westinghouse Electric Corporation
ATTN: Henry P. Kalapaca, M.S. 3525

Official Record Copy, Mr. K. D. Smith, AFWL/DYV